

Laporan Tugas Besar 3 IF2211 Strategi Algoritma

Semester II tahun 2024/2025

**Pemanfaatan *Pattern Matching* untuk Membangun Sistem ATS (*Applicant Tracking System*)
Berbasis CV Digital**



Disusun oleh:

William Andrian (13523006)

Abdullah Farhan (13523042)

Sebastian Enrico Nathanael (13523134)

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

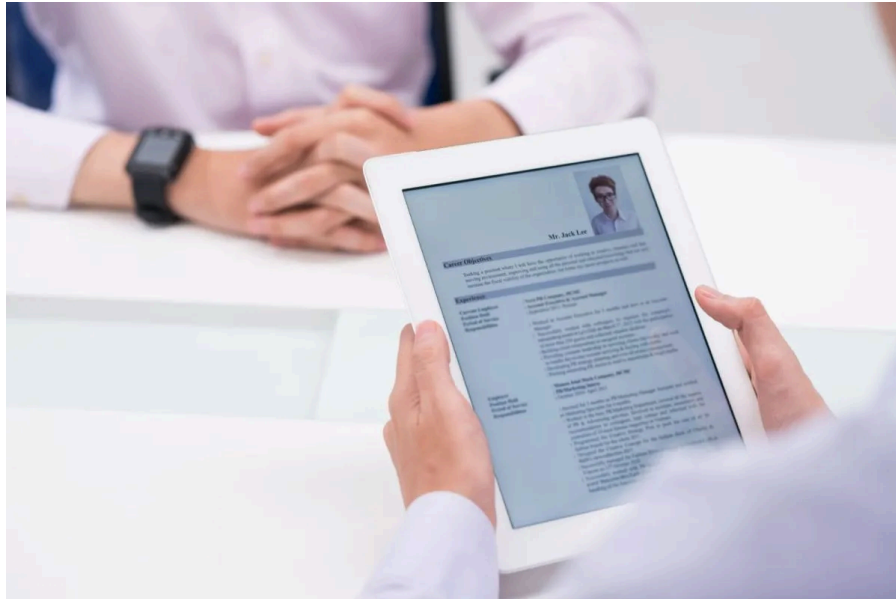
2025

Daftar Isi

BAB 1	4
Deskripsi Tugas	4
Penjelasan Implementasi	5
Penggunaan Program	8
Spesifikasi Wajib	9
Spesifikasi Bonus	11
BAB 2	13
Landasan Teori	13
2.1 Pattern Matching	14
2.2 Algoritma Knuth-Morris-Pratt (KMP)	14
2.3 Algoritma Boyer-Moore (BM)	15
2.4 Algoritma Aho-Corasick	15
2.5 Levenshtein Distance	16
2.6 Regular Expression (Regex)	16
BAB 3	17
Analisis Pemecahan Masalah	17
BAB 4	20
Implementasi dan Pengujian	20
3.1 Implementasi	20
3.1.1 ahocor.py	20
3.1.2 bm.py	24
3.1.3 kmp.py	26
3.1.4 levenshtein.py	28
3.1.5 extract_edu.py	33
3.1.6 extract_exp.py	37
3.1.7 extract_skill.py	42
3.1.8 db.py	46
3.1.9 extract.py	48
3.1.10 pdf_to_text.py	49
3.1.11 seeding.py	51
3.1.12 utils.py	58
3.1.13 main.py	59
3.1.14 Tuning Pemilihan Parameter Default	81
3.2 Pengujian	82
3.2.1 Pengujian KMP	82
BAB 5	85
Kesimpulan, Saran, dan Refleksi	85
5.1 Kesimpulan	85
5.2 Saran	85
5.3 Refleksi	85
Lampiran	86
Daftar Pustaka	87

BAB 1

Deskripsi Tugas



Gambar 1. CV ATS dalam Dunia Kerja

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

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

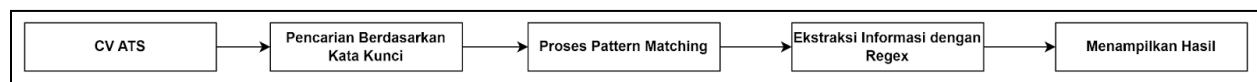
Meskipun demikian, salah satu tantangan besar dalam pengembangan sistem ATS adalah kemampuan untuk memproses dokumen CV dalam format PDF yang tidak selalu terstruktur. Dokumen seperti ini memerlukan metode canggih untuk mengekstrak informasi penting seperti identitas, pengalaman kerja, keahlian, dan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

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

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

Penjelasan Implementasi

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



Gambar 2. Skema Implementasi *Applicant Tracking System*

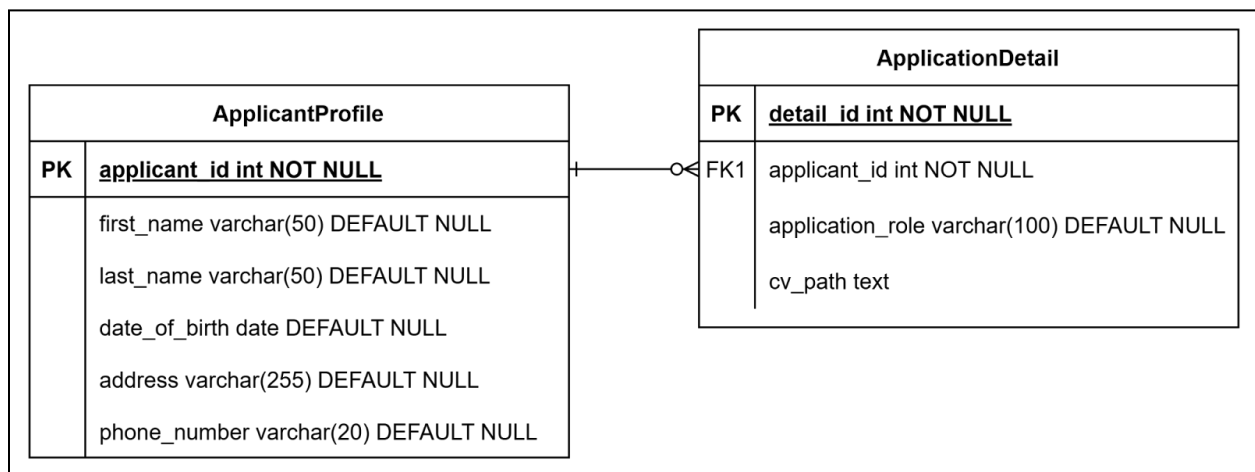
Sistem ini bertujuan untuk mencocokkan kata kunci dari user terhadap isi CV pelamar kerja dengan pendekatan pattern matching menggunakan algoritma KMP (Knuth-Morris-Pratt) atau BM (Boyer-Moore). Semua proses dilakukan secara in-memory, tanpa menyimpan hasil pencarian—hanya data mentah (raw) CV yang disimpan. Pengguna (HR atau rekruter) akan memberikan input berupa daftar kata kunci yang ingin dicari (misalnya: "python", "react", dan "sql") serta jumlah CV yang ingin ditampilkan (misalnya Top 10 matches). Setiap file CV dalam format PDF akan dikonversi menjadi satu string panjang yang memuat seluruh teks dari dokumen tersebut. Proses konversi ini bertujuan untuk mempermudah pencocokan pola menggunakan algoritma string matching, sehingga setiap keyword dapat dicari secara efisien dalam satu representasi data linear.

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

Tabel 1. Hasil ekstraksi teks dari CV ATS

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

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



Gambar 3. Skema basis data CV ATS

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

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

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

Penggunaan Program

The screenshot displays the 'CV Analyzer App' interface. At the top, there's a 'Keywords' input field containing 'React, Express, HTML'. Below it, the 'Search Algorithm' is set to 'BM' (with 'KMP' as an alternative). A 'Top Matches' dropdown is set to '3'. A 'Search' button is located below these controls. The 'Results' section shows '100 CVs scanned in 100ms' and lists three candidates: Farhan (4 matches), Aland (1 match), and Ariel (1 match). Each candidate's entry includes a list of matched keywords and their occurrences, along with 'Summary' and 'View CV' buttons.

Candidate	Matches	Matched keywords
Farhan	4 matches	1. React: 1 occurrence 2. Express: 2 occurrences 3. HTML: 1 occurrence
Aland	1 match	1. React: 1 occurrence
Ariel	1 match	1. Express: 1 occurrence

Gambar 4. Contoh Antarmuka Program (Halaman *Home*)

CV Summary

Farhan
 Birthdate: 05-19-2025
 Address: Masjid Salman ITB
 Phone: 0812 3456 7890

Skills:
 React Express HTML

Job History:
CTO
 2003-2004
 Leading the organization's technology strategies

Education:
Informatics Engineering (Institut Teknologi Bandung)
 2022-2026

Gambar 5. Contoh Antarmuka Program (Halaman *Summary*)

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

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

Secara umum, berikut adalah cara umum penggunaan program:

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

Spesifikasi Wajib

Pada Tugas Besar ini, buatlah sebuah sistem yang dapat melakukan pencocokan dan pencarian informasi pelamar kerja berdasarkan [dataset CV ATS Digital](#). Data yang digunakan merupakan gabungan **20 data pertama dari setiap *category/bidang*, yang telah terurut secara leksikografis** (i.e. 20 data dari *category* HR + 20 data dari *category* Designer, dst). Sistem harus memiliki fitur dengan detail sebagai berikut:

1. Sistem yang dibangun pada tugas besar ini bertujuan untuk melakukan pencocokan dan pencarian data pelamar kerja berbasis CV yang diunggah oleh pengguna. Sistem dikembangkan menggunakan **bahasa pemrograman Python** dengan **antarmuka desktop** berbasis pustaka seperti **Tkinter, PyQt, atau framework lain** yang relevan. Dalam proses pencocokan kata kunci, sistem wajib mengimplementasikan algoritma **Knuth-Morris-Pratt (KMP)** dan **Boyer-Moore (BM)**. Untuk mengukur kemiripan saat terjadi kesalahan input atau perbedaan penulisan, sistem juga menerapkan algoritma **Levenshtein Distance**. Selain itu, **Regular Expression (Regex)** digunakan untuk mengekstrak informasi penting dari teks CV secara otomatis. Oleh karena itu, penguasaan pengembangan GUI dan pemrosesan string di Python sangat penting untuk menyelesaikan tugas ini secara optimal.
2. Program yang dikembangkan harus menggunakan basis data berbasis **MySQL** untuk menyimpan informasi hasil ekstraksi dari CV yang telah diunggah. Basis data akan menyimpan informasi berupa profil pelamar beserta lokasi penyimpanan file CV di dalam sistem.
3. Fitur utama dari sistem ini adalah kemampuannya untuk ekstraksi teks dari CV dalam format PDF. Setelah dokumen CV diunggah, program harus mampu melakukan ekstraksi teks secara otomatis dan mengubahnya menjadi profil pelamar kerja. Profil tersebut akan ditampilkan kepada pengguna tanpa perlu intervensi manual tambahan. Proses ini akan membantu mempercepat identifikasi dan penilaian awal terhadap pelamar.

4. Sistem wajib menyediakan fitur **pencarian terhadap data pelamar** menggunakan **kata kunci** atau kriteria tertentu yang ditentukan oleh pengguna (misalnya nama, skill tertentu, atau pengalaman kerja). Pencarian ini akan dilakukan terhadap semua data dalam database dan bertujuan untuk menemukan pelamar yang paling relevan dengan kriteria pencarian tersebut. Proses pencarian dilakukan sepenuhnya secara in-memory agar hasilnya cepat dan responsif. Proses pencarian utamanya dilakukan secara *exact matching*.
5. Setelah *exact matching*, apabila tidak ditemukan kecocokan secara persis, sistem harus melakukan *fuzzy matching*. Untuk setiap kata kunci yang tidak ditemukan satupun kemunculan saat exact matching, lakukan pencarian kembali dengan **perhitungan tingkat kemiripan menggunakan algoritma Levenshtein Distance**. Algoritma ini memungkinkan sistem untuk tetap menampilkan hasil pencarian yang relevan, meskipun terdapat perbedaan minor atau kesalahan ketik pada input pengguna. Hal ini sangat membantu pengguna untuk tetap mendapatkan hasil terbaik tanpa harus memasukkan kata kunci secara sempurna.
6. Apabila salah satu hasil pencarian di-klik, sistem harus dapat menampilkan **ringkasan/summary** dari lamaran tersebut. Pada halaman ringkasan, harus terdapat opsi (e.g. tombol) untuk **melihat CV secara keseluruhan**.
7. Informasi yang ditampilkan dalam **ringkasan/summary** CV dari hasil pencarian harus mencakup **data penting dari pelamar**, yaitu identitas (nama, kontak, dan informasi pribadi lainnya) yang diperoleh dari basis data. Kemudian terdapat beberapa data yang diperoleh dengan cara ekstraksi melalui **regular expression**, meliputi:
 - Ringkasan pelamar (summary/overview)
 - Keahlian pelamar (skill)
 - Pengalaman kerja (e.g. tanggal dan jabatan)
 - Riwayat pendidikan (e.g. tanggal kelulusan, universitas, dan gelar)

Dengan menampilkan informasi-informasi penting tersebut, sistem dapat memberikan ringkasan profil yang relevan kepada pengguna.

8. Pengguna aplikasi dapat memilih algoritma pencocokan yang ingin digunakan untuk *exact matching*, yaitu antara **KMP** atau **BM**, (bisa pula **Aho-Corasick** apabila mengerjakan **bonus**), sebelum memulai proses pencarian. Pilihan algoritma ini akan mempengaruhi cara sistem memindai dan mencocokkan kata kunci dengan isi CV. Hal ini memberikan fleksibilitas dan pemahaman algoritmik yang lebih luas bagi pengguna atau pengembang.

9. Pengguna aplikasi dapat **menentukan jumlah CV yang ditampilkan**. CV yang ditampilkan diurutkan mulai dari CV dengan jumlah kecocokan kata kunci terbanyak.
10. Setelah pencarian CV, sistem akan **menampilkan waktu pencarian**. Terdapat **2 waktu berbeda** yang perlu ditampilkan. Pertama adalah waktu pencarian ***exact match*** menggunakan algoritma KMP atau BM. Kemudian, tampilkan juga waktu pencarian ***fuzzy match*** menggunakan Levenshtein Distance apabila terdapat kata kunci yang belum ditemukan.
11. Aplikasi yang dibuat harus memiliki **antarmuka pengguna (user interface) yang intuitif dan menarik**, sehingga mudah digunakan bahkan oleh pengguna awam. Komponen-komponen penting seperti, input *keyword*, pemilihan algoritma, serta hasil pencarian harus disusun dengan jelas dan rapi. Pengembang juga diperkenankan menambahkan fitur tambahan yang dapat memperkaya fungsi dan pengalaman pengguna, sebagai bentuk kreativitas dan inisiatif dalam mengembangkan sistem yang lebih bermanfaat dan inovatif.

Spesifikasi Bonus

Bagian ini hanya boleh dikerjakan apabila seluruh spesifikasi wajib dari Tugas Besar telah berhasil dipenuhi. Pengerjaan bagian bonus bersifat opsional, namun semakin banyak bagian bonus yang dikerjakan, maka semakin tinggi tambahan nilai yang bisa diperoleh.

1. Enkripsi Data Profil Applicant (maksimal 5 poin)

Lakukan proses enkripsi terhadap data profil applicant yang disimpan di dalam basis data untuk menjaga kerahasiaan informasi pribadi pelamar kerja. Enkripsi ini perlu diterapkan agar data tetap aman meskipun terjadi akses langsung ke basis data. Implementasi **tidak diperkenankan menggunakan pustaka enkripsi bawaan Python** (cryptography atau hashlib), **semakin kompleks dan aman skema enkripsi yang dibuat maka potensi nilai bonus pun akan semakin tinggi**.

NOTES: Data yang disediakan untuk keperluan demo tidak menggunakan enkripsi

2. Implementasi Algoritma Aho-Corasick (maksimal 10 poin)

Tambahkan dukungan algoritma Aho-Corasick sebagai alternatif metode pencarian kata kunci yang efisien untuk multi-pattern matching. Dengan algoritma ini, sistem dapat

mencocokkan seluruh daftar keyword sekaligus dalam satu proses traversal teks, sehingga lebih cepat dibandingkan pendekatan konvensional satu per satu. Kehadiran opsi algoritma ini menunjukkan pemahaman lanjutan terhadap strategi optimasi pencarian string.

3. Pembuatan Video Aplikasi (maksimal 5 poin)

Buatlah video presentasi mengenai aplikasi yang telah dikembangkan, mencakup penjelasan fitur-fitur utama serta demonstrasi penggunaannya. Video harus menyertakan audio narasi dan menampilkan wajah dari seluruh anggota kelompok. Kualitas penyampaian dan visual akan mempengaruhi penilaian. Upload video ke YouTube dan pastikan video dapat diakses publik. Sebagai referensi, Anda dapat melihat video tugas besar dari tahun-tahun sebelumnya dengan kata kunci seperti “Tubes Stima”, “Tugas Besar Stima”, atau “Strategi Algoritma”.

BAB 2

Landasan Teori

Pada Tugas Besar kali ini kita diminta untuk membuat sistem *Applicant Tracking System* (ATS) berbasis desktop/web dirancang untuk membantu proses seleksi pelamar kerja secara otomatis melalui pencocokan informasi pada dokumen CV digital. Sistem ini mengimplementasikan berbagai algoritma *pattern matching* seperti Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick untuk mencari kemunculan kata kunci yang dimasukkan oleh pengguna dalam teks hasil ekstraksi CV. Tujuan utama aplikasi ini adalah meningkatkan efisiensi rekrutmen dengan memungkinkan HR atau rekruter menemukan kandidat paling relevan dalam waktu yang cepat.

Setiap CV yang diunggah akan dikonversi dari file PDF menjadi teks, kemudian diproses menggunakan algoritma pencocokan yang dipilih. Aplikasi menyediakan fitur pencarian exact match berdasarkan algoritma yang dipilih, serta pencarian fuzzy menggunakan algoritma Levenshtein Distance jika tidak ditemukan kecocokan sempurna. Selain itu, aplikasi ini juga menampilkan informasi penting dari CV yang telah diekstraksi menggunakan *regular expression*, seperti ringkasan pelamar, daftar keahlian, pengalaman kerja, dan riwayat pendidikan.

Aplikasi dilengkapi dengan antarmuka pengguna yang intuitif, memungkinkan pengguna untuk memasukkan kata kunci pencarian, memilih algoritma, dan menampilkan daftar CV teratas berdasarkan relevansi. Hasil pencarian disajikan dalam bentuk kartu yang informatif, dengan opsi untuk melihat ringkasan data atau membuka file CV asli. Dengan pendekatan ini, aplikasi tidak hanya menjadi alat bantu teknis, tetapi juga menghadirkan solusi nyata bagi proses rekrutmen modern berbasis data.

2.1 Pattern Matching

Pattern Matching adalah proses pencocokan pola atau urutan karakter tertentu di dalam sebuah teks. Dalam konteks rekayasa perangkat lunak dan pemrosesan dokumen, pattern matching digunakan untuk mendeteksi apakah sebuah string (*pattern*) muncul dalam string lain yang lebih besar (*text*). Teknik ini menjadi dasar dalam berbagai aplikasi seperti pencarian informasi, pengolahan bahasa alami, serta sistem pelacakan dokumen digital seperti *Applicant Tracking System* (ATS).

Pada sistem ATS, pattern matching digunakan untuk mencari keberadaan kata kunci tertentu yang dimasukkan oleh pengguna (HR) dalam dokumen CV pelamar. Karena CV

umumnya berupa teks tidak terstruktur dalam format PDF, maka pattern matching memberikan metode yang efisien untuk menelusuri dan mengekstraksi informasi secara otomatis. Penggunaan algoritma pattern matching memungkinkan sistem untuk bekerja secara cepat meskipun harus memproses ratusan hingga ribuan CV.

2.2 Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah salah satu algoritma pencocokan string yang dirancang untuk menghindari pemeriksaan karakter yang sudah pernah dibandingkan sebelumnya. Inti dari algoritma ini terletak pada preprocessing pola yang menghasilkan sebuah array prefix function atau failure function. Array ini digunakan untuk menentukan seberapa jauh pola dapat digeser setelah terjadi mismatch, tanpa perlu mengulangi perbandingan dari awal.

KMP memiliki kompleksitas waktu $O(n + m)$, di mana n adalah panjang teks dan m adalah panjang pola. Karena efisiensi ini, algoritma KMP sangat sesuai untuk digunakan pada proses pencarian kata kunci dalam CV digital yang telah dikonversi menjadi string panjang. Dengan implementasi yang tepat, KMP dapat memberikan hasil pencarian yang akurat dalam waktu yang singkat, membuat sistem ATS lebih responsif.

Kompleksitas Waktu :

$$O(n + m)$$

Preprocessing pola: $O(m)$

Matching proses terhadap teks: $O(n)$

n = panjang teks

m = panjang pola

Langkah Langkahnya :

1. Buat prefix table
2. Iterasi teks dan pola. Jika cocok, lanjutkan. Jika tidak cocok, geser pola sesuai nilai dari prefix table tanpa mengulang pencocokan dari awal.

2.3 Algoritma Boyer-Moore (BM)

Boyer-Moore adalah algoritma pencocokan string yang terkenal dengan efisiensinya dalam praktik karena melakukan pencocokan dari kanan ke kiri pola. Algoritma ini

menggunakan dua heuristik utama, yaitu *bad character rule* dan *good suffix rule*, untuk menentukan seberapa jauh pola dapat digeser saat terjadi mismatch. Hal ini memungkinkan Boyer-Moore untuk melewati sebagian besar karakter dalam teks yang tidak relevan dengan pencarian.

Keunggulan Boyer-Moore terletak pada kemampuannya untuk melakukan lompatan yang besar dalam teks, membuatnya sangat cepat ketika pola jarang muncul dalam teks. Dalam konteks sistem ATS, algoritma ini efektif untuk memindai CV yang panjang dengan cepat dan hanya memfokuskan pencarian pada bagian yang potensial relevan dengan keyword. Karena itulah, Boyer-Moore sering menjadi pilihan utama dalam aplikasi real-world seperti search engine dan document scanning.

Kompleksitas Waktu :

Worst case: $O(n \times m)$

Average case: $O(n / m)$

2.4 Algoritma Aho-Corasick

Aho-Corasick adalah algoritma pencocokan multi-pattern yang memungkinkan pencarian banyak pola sekaligus dalam satu traversal teks. Algoritma ini membangun sebuah struktur trie dari kumpulan pola yang ingin dicari, lalu melengkapinya dengan *failure links* seperti pada KMP. Dengan menggunakan struktur ini, proses pencocokan semua pola dilakukan dalam kompleksitas waktu linear terhadap panjang teks dan total panjang semua pola.

Dalam sistem ATS, Aho-Corasick sangat bermanfaat jika pengguna ingin mencari banyak keyword sekaligus, karena performanya jauh lebih baik dibanding pencarian satu per satu menggunakan KMP atau BM. Dengan Aho-Corasick, sistem dapat mencocokkan ratusan keyword secara serentak, mengurangi waktu eksekusi secara signifikan, terutama saat memindai banyak CV. Namun, waktu pembuatan trie yang diperlukan di awal mungkin mengalahkan waktu pencarian yang terhemat jika *pattern* yang diberikan sedikit atau teks yang diberikan kurang panjang. Sehingga algoritma ini paling optimal ketika mendapatkan banyak *pattern* sebagai masukannya.

2.5 Levenshtein Distance

Levenshtein Distance adalah algoritma untuk mengukur jarak edit antara dua string, yaitu jumlah minimum operasi penyisipan, penghapusan, atau substitusi karakter yang diperlukan untuk mengubah satu string menjadi string lainnya. Algoritma ini digunakan dalam *fuzzy matching*, di mana sistem tetap mampu menemukan kecocokan meskipun terdapat perbedaan kecil dalam penulisan.

Dalam implementasi sistem ATS, Levenshtein Distance menjadi komponen penting ketika tidak ditemukan kecocokan sempurna melalui KMP atau BM. Misalnya, jika pengguna salah ketik kata "JavaScrip" alih-alih "JavaScript", maka algoritma ini tetap dapat mendeteksi kemiripan dan menampilkan hasil relevan. Keberadaan fitur ini meningkatkan fleksibilitas dan user experience dari sistem ATS, karena tidak mengharuskan pencarian yang 100% presisi.

2.6 Regular Expression (Regex)

Regular Expression (Regex) adalah metode pencocokan pola berbasis ekspresi simbolik yang digunakan untuk mengekstrak atau memvalidasi string tertentu dari teks yang tidak terstruktur. Regex sangat kuat karena mampu mengenali berbagai format data seperti tanggal, email, nomor telepon, nama institusi, dan pola teks lainnya melalui aturan simbol yang singkat dan fleksibel.

Dalam konteks sistem ATS, Regex digunakan untuk mengekstrak informasi penting dari isi CV digital, seperti ringkasan pelamar, daftar keahlian (skills), pengalaman kerja, dan riwayat pendidikan. Dengan menggunakan Regex, proses ekstraksi dapat dilakukan secara otomatis tanpa perlu struktur yang konsisten dalam dokumen. Hal ini sangat penting karena CV dari berbagai pelamar memiliki format yang sangat bervariasi dan tidak selalu mengikuti standar baku.

BAB 3

Analisis Pemecahan Masalah

3.1 Langkah-Langkah Pemecahan Masalah

Dalam pengembangan sistem *Applicant Tracking System* (ATS) berbasis CV digital, permasalahan utama yang dihadapi adalah bagaimana mencari dan mencocokkan kata kunci relevan secara cepat dan efisien dari ratusan dokumen tidak terstruktur. Untuk itu, kami mengadopsi pendekatan berbasis *pattern matching* dengan algoritma KMP, Boyer-Moore, dan Aho-Corasick, serta teknik fuzzy matching dengan Levenshtein Distance sebagai fallback.

Langkah-langkah utama dalam menyelesaikan masalah tersebut adalah:

1. Melakukan ekstraksi teks dari CV digital dalam format PDF dan mengubahnya menjadi representasi string linear.
2. Menyediakan antarmuka input bagi user (HR) untuk memasukkan kata kunci dan memilih algoritma pencarian.
3. Menerapkan algoritma pencocokan pola (KMP, BM, atau Aho-Corasick) untuk mencari keberadaan kata kunci dalam setiap dokumen.
4. Menampilkan hasil pencarian sesuai relevansi dengan visualisasi informasi ringkas dari setiap CV.
5. Jika tidak ditemukan hasil dari exact match, dilakukan fuzzy match dengan Levenshtein Distance.

3.2 Proses Pemetaan Masalah

Knuth-Morris-Pratt (KMP)

- Masalah: Mencari satu keyword dalam teks CV secara cepat tanpa perbandingan ulang.
- Pemetaan:
 - Pola = keyword yang dimasukkan pengguna.
 - Teks = hasil ekstraksi dari dokumen CV.

- *Prefix function* digunakan untuk menghindari perbandingan ulang jika terjadi mismatch.
- Strategi: Digunakan ketika user ingin mencari satu per satu keyword (per kata) dan memastikan pencarian berjalan linear terhadap ukuran teks.

Boyer-Moore (BM)

- Masalah: Mempercepat pencarian kata kunci dalam teks panjang, dengan optimisasi lompatan saat mismatch.
- Pemetaan:
 - Pola = keyword.
 - Teks = isi CV.
 - Heuristik bad character dan good suffix untuk mengatur seberapa jauh pola dapat digeser.
- Strategi: Digunakan ketika teks CV sangat panjang dan ingin mengurangi jumlah perbandingan karakter secara signifikan.

Aho-Corasick

- Masalah: Mencari banyak kata kunci sekaligus dalam satu traversal teks.
- Pemetaan:
 - Pola = seluruh daftar kata kunci dari input user.
 - Teks = seluruh isi CV.
 - Struktur trie dan failure link dibangun sebelum pencarian.
- Strategi: Cocok ketika jumlah keyword sangat banyak (≥ 5) dan membutuhkan efisiensi tinggi dalam satu kali scanning teks.

3.3 Fungsionalitas dan Arsitektur Perangkat Lunak

Fitur Fungsional :

1. Ekstraksi Teks PDF
Konversi dokumen CV dalam format PDF ke string menggunakan pustaka seperti pypdf atau pdfplumber.
2. Input Kata Kunci dan Pilihan Algoritma
Pengguna memasukkan kata kunci, memilih algoritma (KMP, BM, atau Aho-Corasick), dan menentukan jumlah hasil.

3. Pencarian dan Pencocokan Pola
Sistem melakukan exact matching terhadap CV menggunakan algoritma yang dipilih.
4. Fallback Fuzzy Matching (Levenshtein)
Jika exact match gagal, sistem melakukan fuzzy match berdasarkan ambang batas kemiripan.
5. Hasil Pencarian Terurut
CV ditampilkan berdasarkan jumlah keyword yang cocok atau tingkat kemiripan tertinggi.
6. Tampilan Ringkasan & File CV
Menyediakan tombol “Summary” (menampilkan hasil ekstraksi) dan “View CV” (menampilkan file PDF asli).

Arsitektur Aplikasi

- Frontend: Flet (desktop Python UI)
- Backend:
 - Ekstraksi teks dari pdf ke text : PyMuPDF
 - DBMS: MySQL menyimpan data pelamar dan jalur file CV
 - Python

3.4 Contoh Ilustrasi Kasus

Kasus:

Seorang HR ingin mencari pelamar yang memiliki keahlian di bidang “React”, “Express”, dan “MongoDB”. Namun ia kesusahan untuk melihat satu persatu file cv para pendaftar. Ada 480 cv yang ia terima. HR mengingat bahwa ia pernah membeli suatu software ATS, ia akan memanfaatkan software pattern matching.

Langkah Pencarian Solusi:

1. HR memasukkan tiga keyword tersebut.
2. HR memilih apakah akan menggunakan KMP, BM, atau Aho-Corasick.
3. Jika KMP/BM dipilih:
 - Sistem melakukan pencarian satu per satu terhadap 3 keyword dalam seluruh CV.
 - Menghitung frekuensi kemunculan.

4. Jika Aho-Corasick dipilih:
 - Ketiga keyword dimasukkan ke dalam trie.
 - Teks CV ditraversal satu kali, sistem mendeteksi semua match sekaligus.
5. Hasil disortir berdasarkan jumlah kecocokan (match count), dan 10 teratas ditampilkan.
6. Jika satu keyword tidak ditemukan, sistem menghitung kemiripannya (misalnya "MngoDB" → "MongoDB") menggunakan Levenshtein Distance.
7. HR dapat klik tombol "Summary" untuk melihat informasi seperti:
 - Ringkasan pelamar
 - Daftar skill
 - Pengalaman kerja
 - Pendidikan

Serta ia dapat melihat langsung cv-nya

BAB 4

Implementasi dan Pengujian

3.1 Implementasi

3.1.1 ahocor.py

```
# Aho-Corasick String Matching Algorithm

from collections import deque

class AhoCorasick:

    def __init__(self, words: list[str]):

        self.root = TrieNode()

        for word in words:

            self.root.add_word(word)

        self.build_failure_links()

    def build_failure_links(self):

        self.root.fail_link = self.root

        queue: deque[TrieNode] = deque()

        for child in self.root.children.values():

            child.fail_link = self.root

            queue.append(child)

        while queue:

            curr_node = queue.popleft()
```

```

        for char, child in curr_node.children.items():

            queue.append(child)

            curr_fail_link = curr_node.fail_link

            while char not in curr_fail_link.children.keys() and curr_fail_link
!=self.root:

                curr_fail_link = curr_fail_link.fail_link

            child.fail_link = curr_fail_link.children.get(char, self.root)

            child.output += child.fail_link.output

    @staticmethod

    def search(text:str, words:list[str]) -> list[tuple[int, str]]:

        """

        Searches for all occurrences of words in the text.

        Returns a list of tuples (index, word).

        """

        ac = AhoCorasick(words)

        results = []

        curr_node = ac.root

        for i, char in enumerate(text):

            if char in curr_node.children.keys():

                curr_node = curr_node.children[char]

                if curr_node.output:

                    for word in curr_node.output:

                        results.append((i - len(word)+1, word))

            elif curr_node == ac.root:

```

```

        continue

    else:

        curr_node = curr_node.fail_link

    return results

class TrieNode:

    def __init__(self):

        self.children:dict[str,TrieNode] = {}

        self.fail_link:TrieNode = None

        self.output = []

    def add_word(self,word:str):

        """

        Adds a word to the TrieNode.

        """

        node = self

        for char in word:

            if char not in node.children:

                node.children[char] = TrieNode()

            node = node.children[char]

        node.output.append(word)

```

<code>__init__(self, words: list[str])</code>	Konstruktor dari kelas AhoCorasick, Menerima daftar kata yang ingin dicari dalam teks, Membangun struktur Trie dari daftar kata tersebut. Memanggil <code>build_failure_links()</code> untuk membangun <i>failure link</i> yang penting dalam pencarian.
<code>build_failure_links(self)</code>	Membangun <i>failure links</i> untuk setiap node dalam trie. <i>Failure link</i> digunakan untuk fallback (kembali ke state sebelumnya) ketika terjadi mismatch saat traversal teks.
<code>@staticmethod def search(text:str, words:list[str]) -> list[tuple[int, str]]</code>	Melakukan pencarian semua words dalam text. Merupakan fungsi utama yang mengembalikan hasil pencocokan dalam bentuk: List of tuples (posisi_awal_kemunculan, kata)
<code>class TrieNode</code> <code>__init__(self)</code>	Kelas yang mewakili satu simpul (node) dalam Trie.
<code>add_word(self, word:str)</code>	Menambahkan satu word ke dalam trie. Setiap karakter dari word menjadi edge ke node baru jika belum ada. Setelah selesai, word dimasukkan ke daftar output dari node akhir (menandakan akhir kata ditemukan di node itu).

3.1.2 bm.py

```
# Boyer Moore String Search Algorithm

def build_last_occurrence(pattern:str, charset=None) -> dict:

    """

    Builds last occurrence table for 'pattern'.

    returns a map of char and their last index position.

    """

    if charset is None:

        charset = set(pattern)

    last_occurrence = {char: -1 for char in charset}

    for i, char in enumerate(pattern):

        last_occurrence[char] = i

    return last_occurrence


def boyer_moore_search(text: str, pattern: str) -> list[int]:

    """

    Finds all occurrences of pattern in text.

    Returns a list of start indexes.

    """

    n, m = len(text), len(pattern)

    if m == 0:

        return []
```



```

last_occurrence = build_last_occurrence(pattern)

results = []

s = 0 # shift

while s <= n - m:

    j = m - 1 # index of last char in pattern

    while j >= 0 and pattern[j] == text[s + j]:

        j -= 1

    if j < 0:

        results.append(s)

        s += (s + m < n) and (m - last_occurrence.get(text[s + m], -1)) or 1

    else:

        s += max(1, j - last_occurrence.get(text[s + j], -1))

return results

```

build_last_occurrence(pattern: str, charset=None) -> dict	Membangun tabel kemunculan terakhir (last occurrence table) untuk setiap karakter dalam pola (<i>pattern</i>). Fungsi ini mengembalikan dictionary yang memetakan karakter ke indeks terakhirnya dalam pattern.
boyer_moore_search(text: str, pattern: str) -> list[int]	Fungsi utama untuk melakukan pencarian semua kemunculan pattern dalam text menggunakan algoritma Boyer-Moore. Mengembalikan daftar indeks kemunculan pattern pada text.

	<p>Memfaatkan hasil <code>build_last_occurrence()</code> untuk menentukan pergeseran (shift) saat terjadi mismatch.</p> <p>Pencocokan dimulai dari karakter paling kanan dalam pattern, bukan dari kiri seperti algoritma naive atau KMP.</p> <p>Saat terjadi mismatch, algoritma menghitung pergeseran optimal berdasarkan posisi karakter yang tidak cocok.</p>
--	---

3.1.3 kmp.py

```
# file: src/kmp_utils.py

def compute_lps(pattern: str) -> list[int]:
    """
    Bangun array LPS (longest proper prefix which is also suffix)
    untuk pola 'pattern'.
    """
    m = len(pattern)

    lps = [0] * m

    length = 0 # panjang prefix-suffix saat ini

    i = 1

    while i < m:
```

```

        if pattern[i] == pattern[length]:

            length += 1

            lps[i] = length

            i += 1

        else:

            if length != 0:

                # mundur ke nilai LPS sebelumnya

                length = lps[length - 1]

            else:

                lps[i] = 0

                i += 1

    return lps

def kmp_search(text: str, pattern: str) -> list[int]:

    """

    Cari semua kemunculan 'pattern' di 'text' menggunakan KMP.

    Kembalikan list posisi (0-based) di mana pattern mulai cocok.

    """

    n, m = len(text), len(pattern)

    if m == 0:

        return []

    lps = compute_lps(pattern)

    results = []

    i = j = 0 # i untuk text, j untuk pattern

```

```

while i < n:

    if text[i] == pattern[j]:

        i += 1

        j += 1

        if j == m:

            # ketemu match berakhir di i-1, maka start = i-m

            results.append(i - m)

            j = lps[j - 1]

        else:

            if j != 0:

                j = lps[j - 1]

            else:

                i += 1

return results

```

compute_lps(pattern: str) -> list[int]

Membangun LPS array (Longest Proper Prefix which is also Suffix) untuk pattern. LPS digunakan untuk menentukan berapa jauh pattern bisa digeser ketika terjadi mismatch, tanpa perlu membandingkan kembali karakter yang sudah dicocokkan sebelumnya. lps[i] menyimpan panjang proper prefix dari pattern[0..i] yang juga merupakan suffix.

Digunakan dalam proses pencocokan untuk menghindari pencocokan ulang karakter

	Kompleksitas waktu: $O(m)$ di mana m adalah panjang pattern.
<code>kmp_search(text: str, pattern: str) -> list[int]</code>	<p>Melakukan pencarian semua kemunculan pattern dalam text menggunakan algoritma KMP.</p> <p>Mengembalikan list posisi (0-based) di mana pattern mulai cocok dalam text. Menggunakan LPS untuk menentukan berapa banyak karakter dalam pattern yang bisa dilewati saat mismatch. Kompleksitas waktu: $O(n + m)$ di mana $n = \text{len}(\text{text})$ dan $m = \text{len}(\text{pattern})$.</p>

3.1.4 levenshtein.py

```
import math

def levenshtein_distance(a: str, b: str) -> int:
    """
    Menghitung jarak Levenshtein antara dua string.
    """
    len_a, len_b = len(a), len(b)

    # Buat matriks ukuran (len_a+1) x (len_b+1)
    dp = [[0 for _ in range(len_b + 1)] for _ in range(len_a + 1)]

    # Inisialisasi baris dan kolom pertama

    for i in range(len_a + 1):
```

```

        dp[i][0] = i # Biaya menghapus semua karakter dari a

    for j in range(len_b + 1):

        dp[0][j] = j # Biaya menambahkan semua karakter ke a

# Isi matriks

for i in range(1, len_a + 1):

    for j in range(1, len_b + 1):

        if a[i - 1] == b[j - 1]:

            cost = 0 # karakter sama, tidak ada biaya

        else:

            cost = 1 # karakter beda, ada biaya substitusi

        dp[i][j] = min(

            dp[i - 1][j] + 1,      # Hapus

            dp[i][j - 1] + 1,      # Tambah

            dp[i - 1][j - 1] + cost # Substitusi

        )

    return dp[len_a][len_b]

def fuzzy_text_search(text: str, target: str, similarity_threshold: float = 0.125,
max_distance: int = 4) -> tuple[int, list[str]]:

    # Hitung ambang batas (edit_distance_limit) secara dinamis

    target_length = len(target)

```

```

    if target_length == 0:

        return 0, []

    edit_distance_limit = min(math.ceil(similarity_threshold * target_length),
max_distance)

    word_list = text.split()

    match_count = 0

    found_matches = []

    target_lowercase = target.lower()

    for current_word in word_list:

        # edit_distance_limit sebagai threshold dinamis

        if levenshtein_distance(current_word.lower(), target_lowercase) <=
edit_distance_limit:

            match_count += 1

            found_matches.append(current_word)

    return match_count, found_matches

def tune_threshold():

    """

    Simple threshold tuning function

    """

    # Test keywords untuk tuning

    test_keywords = ["python", "javascript", "react", "html", "css", "java", "sql"]

    # Test dengan typos yang umum terjadi

    test_cases = [

```

```

("python", ["pyton", "phyton", "pythn"]), # Missing/extra letters

("javascript", ["javascript", "jadasript", "javascrip"]), # Common typos

("react", ["reac", "reactt", "reat"]), # Missing/extra letters

("html", ["htm", "htlm", "html1"]), # Common typos

]

threshold_candidates = [0.125, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40]

print("🔑 THRESHOLD TUNING RESULTS:")

print("=" * 60)

best_threshold = 0.25

best_score = 0

for threshold in threshold_candidates:

    correct_matches = 0

    total_tests = 0

    for correct_word, typos in test_cases:

        for typo in typos:

            total_tests += 1

            # Simulasi text yang mengandung typo

            test_text = f"experienced in {typo} programming"

            count, matches = fuzzy_text_search(test_text, correct_word, threshold)

            if count > 0: # Jika berhasil match

```



```

        correct_matches += 1

    accuracy = (correct_matches / total_tests) * 100

    print(f"Threshold {threshold:.2f}: {correct_matches}/{total_tests} matches
({accuracy:.1f}% accuracy)")

    if accuracy > best_score:

        best_score = accuracy

        best_threshold = threshold

print("=" * 60)

print(f"🎯 OPTIMAL THRESHOLD: {best_threshold} (Accuracy: {best_score:.1f}%)")

print("=" * 60)

return best_threshold

if __name__ == "__main__":

    # Jalankan tuning threshold

    optimal_threshold = tune_threshold()

    print(f"Optimal threshold untuk dynamicLevenshteinSearch: {optimal_threshold}")

# Dummy data sesuai SQL schema

```


3.1.5 extract_edu.py

```
import re

from typing import List

import fitz  # PyMuPDF

def extract_text_from_pdf(pdf_path: str) -> str:

    text = ""

    with fitz.open(pdf_path) as doc:

        for page in doc:

            try:

                text += page.get_text()

            except AttributeError:

                text += page.getText()

    return text

def extract_education_section(text):

    """

    Fungsi untuk mengekstrak bagian Education dari teks resume
```

```

Args:

    text (str): Teks lengkap dari resume

Returns:

    str: Bagian education yang ditemukan, atau None jika tidak ada
"""

# Pattern 1: Menangkap dari "Education" hingga section berikutnya atau akhir dokumen
# Menggunakan case-insensitive matching

pattern1 = r'(?i)^education\s*\n(?:.*?) (?:^\w+\s*\n|\Z) '

# Pattern 2: Alternative pattern yang lebih fleksibel
# Menangkap dari Education hingga baris yang dimulai dengan huruf kapital (section baru)

pattern2 = r'(?i)education\s*\n(?: (?:^[A-Z] [A-Za-z\s]+\n) .* \n?)* '

# Pattern 3: Pattern yang menangkap Education dan semua baris setelahnya
# hingga menemukan section baru atau kata kunci tertentu

pattern3 =
r'(?i)education\s*\n(?:.*?) (?:\n(?:experience|skills|certifications|interests|additional\s+
information|professional\s+summary|summary|accomplishments|work history)\s*\n|\Z) '

# Pattern 4: Pattern yang menangkap Education dan semua baris setelahnya
# hingga menemukan section baru atau kata kunci tertentu

pattern4 = r'(?i)education and
training\s*\n(?:.*?) (?:\n(?:experience|skills|certifications|interests|additional\s+informa
tion|professional\s+summary|summary|accomplishments|work history)\s*\n|\Z) '

# Coba pattern pertama

```

```

match = re.search(pattern1, text, re.MULTILINE | re.DOTALL)

if match:

    return match.group(1).strip()

# Coba pattern kedua jika yang pertama tidak berhasil

match = re.search(pattern2, text, re.MULTILINE | re.DOTALL)

if match:

    return match.group(1).strip()

# Coba pattern ketiga sebagai fallback

match = re.search(pattern3, text, re.MULTILINE | re.DOTALL)

if match:

    return match.group(1).strip()

match = re.search(pattern4, text, re.MULTILINE | re.DOTALL)

if match:

    return match.group(1).strip()

return None

def extract_education_simple(text: str) -> str:

    """

    Fungsi sederhana untuk mengekstrak bagian Education

    """

    # Pattern sederhana yang menangkap dari Education hingga section berikutnya

    pattern = r'(?i)education\s*\n(.*?) (?:\n[A-Z] [A-Za-z\s]*\n|\Z) '

```

```

match = re.search(pattern, text, re.MULTILINE | re.DOTALL)

if match:

    return match.group(1).strip()

return None

if __name__ == "__main__":

    pdf_paths = [

        "../data/data/ACCOUNTANT/10554236.pdf",

        "../data/data/ACCOUNTANT/10674770.pdf",

        "../data/data/ACCOUNTANT/11163645.pdf"

    ]

    for path in pdf_paths:

        print(f"\n===== {path} =====")

        text = extract_text_from_pdf(path)

        edu = extract_education_section(text)

        if edu:

            print("Bagian Education ditemukan:")

            print("=" * 50)

            print(edu)

        else:

            print("Bagian Education tidak ditemukan.")

```

3.1.6 extract_exp.py

```
import re

from typing import List

import fitz  # PyMuPDF


def extract_text_from_pdf(pdf_path: str) -> str:

    text = ""

    with fitz.open(pdf_path) as doc:

        for page in doc:

            try:

                text += page.get_text()

            except AttributeError:

                text += page.getText()

    return text


def extract_experience_section(text: str) -> List[str]:

    """

    Ekstrak entri pengalaman kerja berdasarkan pola tanggal kerja,

    menangani job title dan company info yang bisa di baris terpisah atau sama.

    """

    lines = text.split('\n')

    results = []

    # Update regex untuk menangani line breaks dan variasi format

    date_pattern = re.compile(
```

```

r"(January|February|March|April|May|June|July|August|September|October|November|December)
\s+\d{4}\s*\n?\s*to\s*\n?\s*(January|February|March|April|May|June|July|August|September|
October|November|December|Present)\s+\d{4}",

    re.IGNORECASE | re.MULTILINE

)

# Tambahkan pattern untuk format MM/YYYY to MM/YYYY

numeric_date_pattern = re.compile(

    r"\b(0[1-9]|1[0-2])/\d{4}\s*to\s*(0[1-9]|1[0-2])/\d{4}\b",

    re.IGNORECASE

)

# Tambahkan pattern untuk format singkat bulan (Aug 2005 to Aug 2007, Aug 2007 to
Current)

short_month_pattern = re.compile(

r"\b(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)\s+\d{4}\s*to\s*(Jan|Feb|Mar|Apr|May
|Jun|Jul|Aug|Sep|Oct|Nov|Dec|Current)\s*(?:\d{4})?\b",

    re.IGNORECASE

)

# Gabungkan teks untuk menangani pattern yang tersebar di multiple lines

full_text = '\n'.join(lines)

# Cari semua matches dari semua pattern

matches = (list(date_pattern.finditer(full_text)) +

```

```

        list(numeric_date_pattern.finditer(full_text)) +

        list(short_month_pattern.finditer(full_text)))

# Sort matches berdasarkan posisi dalam teks

matches.sort(key=lambda x: x.start())

for match in matches:

    match_start = match.start()

    match_end = match.end()

    lines_before_match = full_text[:match_start].count('\n')

    lines_after_match = full_text[:match_end].count('\n')

    # Ambil tanggal yang sudah ditemukan

    date_info = match.group().replace('\n', ' ').strip()

    title_line = ""

    company_line = ""

    # Cek apakah tanggal berada di tengah baris (ada teks sebelum dan sesudah)

    current_line_start = full_text.rfind('\n', 0, match_start) + 1

    current_line_end = full_text.find('\n', match_end)

    if current_line_end == -1:

        current_line_end = len(full_text)

    current_line = full_text[current_line_start:current_line_end].strip()

```



```

# Ekstrak bagian sebelum dan sesudah tanggal dalam baris yang sama

text_before_date = current_line[:match_start - current_line_start].strip()

text_after_date = current_line[match_end - current_line_start:].strip()


# Prioritas 1: Cek apakah ada teks di baris yang sama dengan tanggal

if text_before_date and len(text_before_date.split()) < 10:

    title_line = text_before_date

elif text_after_date and len(text_after_date.split()) < 10:

    title_line = text_after_date


# Prioritas 2: Jika tidak ada di baris yang sama, cek baris sebelumnya

if not title_line and lines_before_match > 0 and lines_before_match < len(lines):

    prev_line = lines[lines_before_match - 1].strip()

    if len(prev_line.split()) < 10 and prev_line:

        title_line = prev_line


# Prioritas 3: Ambil company info dari baris setelahnya jika tidak ada di baris
yang sama

if not text_after_date and lines_after_match < len(lines):

    next_line = lines[lines_after_match+1].strip()

    if len(next_line.split()) < 10 and next_line:

        company_line = next_line

elif text_after_date and not title_line:

    # Jika text_after_date tidak dijadikan title, maka bisa jadi company

```

```

        if len(text_after_date.split()) < 10:

            company_line = text_after_date

        # Jika ada text_before_date dan text_after_date, gunakan keduanya

        if text_before_date and text_after_date:

            if len(text_before_date.split()) < 10 and len(text_after_date.split()) < 10:

                title_line = text_before_date

                company_line = text_after_date

        # Gabungkan semua komponen

        combined = "\n".join(filter(None, [title_line, date_info, company_line]))

        results.append(combined)

    return results

if __name__ == "__main__":

    pdf_paths = [

        "../data/data/ACCOUNTANT/10554236.pdf",

        "../data/data/ACCOUNTANT/10674770.pdf",

        "../data/data/ACCOUNTANT/11163645.pdf"

    ]

    for path in pdf_paths:

        print(f"\n===== {path} =====")

        text = extract_text_from_pdf(path)

```

```

    experiences = extract_experience_section(text)

    for exp in experiences:

        print(exp)

        print("---")

```

3.1.7 extract_skill.py

```

import re

from typing import List

import fitz  # PyMuPDF


def extract_text_from_pdf(pdf_path: str) -> str:

    text = ""

    with fitz.open(pdf_path) as doc:

        for page in doc:

            try:

                text += page.get_text()

            except AttributeError:

                text += page.getText()

    return text


def extract_skills_from_resume(text):

    """

    Fungsi untuk mengekstrak bagian Skills dari teks resume

```

```

"""

# Pattern 1: Menangkap dari "skills" hingga section berikutnya atau akhir dokumen

# Menggunakan case-insensitive matching

pattern1 = r'(?i)^skills\s*\n(?:.*?) (?:^\w+\s*\n|\Z) '

# Pattern 2: Alternative pattern yang lebih fleksibel

# Menangkap dari skills hingga baris yang dimulai dengan huruf kapital (section baru)

pattern2 = r'(?i)skills\s*\n(?: (?!^[A-Z][A-Za-z\s]+\n) .* \n?)* '

# Pattern 3: Pattern yang menangkap skills dan semua baris setelahnya

# hingga menemukan section baru atau kata kunci tertentu

pattern3 =
r'(?i)skills\s*\n(?:.*?) (?:\n(?:experience|skills|certifications|interests|additional\s+inf
ormation|professional\s+summary|summary|accomplishments|work\s+history|highlights)\s*\n|\
Z) '

# Coba semua pattern

patterns = [pattern1, pattern2, pattern3]

# Coba pattern ketiga sebagai fallback

match = re.search(pattern3, text, re.MULTILINE | re.DOTALL)

if match:

    result = match.group(1).strip()

    print(match.group(1).strip())

```

```

        if not re.search(r'(?i)(?:work\s+history|employment|experience|education)',
result):

            return result

# Coba pattern pertama

match = re.search(pattern1, text, re.MULTILINE | re.DOTALL)

if match:

    result = match.group(1).strip()

    print(match.group(1).strip())

    if not re.search(r'(?i)(?:work\s+history|employment|experience|education)',
result):

        return result

# Coba pattern kedua jika yang pertama tidak berhasil

match = re.search(pattern2, text, re.MULTILINE | re.DOTALL)

if match:

    result = match.group(1).strip()

    print(match.group(1).strip())

    if not re.search(r'(?i)(?:work\s+history|employment|experience|education)',
result):

        return result

return None

def extract_skills_simple(text):

```

```

"""
Fungsi sederhana untuk mengekstrak skills

"""

# Pattern yang lebih komprehensif

pattern = r'(?i)Skills\s*[:\-\]?\s*\n?(.*?) (?:\n(?:[A-Z][a-zA-Z\s]*(?:\n|:)|Additional
Information|$))'

match = re.search(pattern, text, re.MULTILINE | re.DOTALL)

if match:

    return match.group(1).strip()

return None

if __name__ == "__main__":

    pdf_paths = [

        "../data/data/ACCOUNTANT/10554236.pdf",

        "../data/data/ACCOUNTANT/10674770.pdf",

        "../data/data/ACCOUNTANT/11163645.pdf"

    ]

    for path in pdf_paths:

        print(f"\n===== {path} =====")

        text = extract_text_from_pdf(path)

        edu = extract_skills_from_resume(text)

        if edu:

            print("Bagian skills ditemukan:")

```

```
        print("=" * 50)

        print(edu)

    else:

        print("Bagian skills tidak ditemukan.")
```

3.1.8 db.py

```
# file: db.py

import mysql.connector

RESUME_DIRECTORY = "../data/data" # Root folder where roles and CVs are stored

TOTAL_CANDIDATES = 200

DB_CONFIG = {

    'host': "localhost",

    'user': "root",

    'password': "12345",

    'database': "cv_ats_db"

}

# ----- DATABASE SETUP -----

def establish_connection():

    return mysql.connector.connect(**DB_CONFIG)
```

```

def get_applicant_by_cv_filename(filename: str):

    conn = establish_connection()

    cursor = conn.cursor(dictionary=True, buffered=True)

    query = """

    SELECT ap.applicant_id, ap.first_name, ap.last_name, ap.date_of_birth,

           ap.address, ap.phone_number, ad.application_role

    FROM ApplicantProfile ap

    JOIN ApplicationDetail ad ON ap.applicant_id = ad.applicant_id

    WHERE ad.cv_path = %s

    """

    cursor.execute(query, (filename,))

    result = cursor.fetchone()

    cursor.close()

    conn.close()

    return result

if __name__ == "__main__":

    filename = "10554236.pdf" # Ganti sesuai data kamu

    data = get_applicant_by_cv_filename(filename)

    if data:

        print("✅ Data ditemukan:")

        for k, v in data.items():

```



```
        print(f"{k}: {v}")

    else:

        print("✗ Tidak ada data dengan nama file tersebut.")
```

3.1.9 extract.py

```
import os

# Set the path to your main 'cvs' directory
data_directory = '../data/data' # <-- Change this to your actual path

# Traverse each subdirectory in 'cvs'
for subdirectory_name in os.listdir(data_directory):

    subdirectory_path = os.path.join(data_directory, subdirectory_name)

    if os.path.isdir(subdirectory_path):

        # Get list of all PDFs sorted alphabetically

        all_pdf_files = sorted(

            [filename for filename in os.listdir(subdirectory_path) if
            filename.lower().endswith('.pdf')]

        )

        # Keep only the first 20
```

```

for pdf_file_to_remove in all_pdf_files[19:]:

    try:

        os.remove(os.path.join(subdirectory_path, pdf_file_to_remove))

        print(f"Deleted: {os.path.join(subdirectory_path, pdf_file_to_remove)}")

    except Exception as error:

        print(f"Error deleting {pdf_file_to_remove}: {error}")

```

3.1.10 pdf_to_text.py

```

import fitz # PyMuPDF

import os

def extract_text_from_pdf(pdf_path: str) -> str:

    """

    Ekstrak teks dari file PDF menjadi satu string.

    """

    text = ""

    with fitz.open(pdf_path) as doc:

        for page in doc:

            # print(dir(page))

            try:

                # Try the newer method first

```

```

        text += page.get_text()

    except AttributeError:

        # Fall back to older method name

        text += page.getText()

    return text

def load_all_cv_texts(cv_root_folder: str) -> list[dict]:

    """

    Memuat semua file PDF dari folder data/ dan mengubahnya menjadi teks.

    Mengembalikan list of dicts dengan key: 'path', 'role', 'text'.

    """

    all_cv_data = []

    for role in os.listdir(cv_root_folder):

        role_folder = os.path.join(cv_root_folder, role)

        if not os.path.isdir(role_folder):

            continue

        for filename in os.listdir(role_folder):

            if filename.lower().endswith(".pdf"):

                full_path = os.path.join(role_folder, filename)

                extracted = extract_text_from_pdf(full_path)

                all_cv_data.append({

                    "path": full_path,

                    "role": role,

```

```

        "filename": filename,

        "text": extracted

    })

    return all_cv_data

def test_pdf_extraction():

    cv_data = load_all_cv_texts("../data/data") # atau sesuaikan

    for data in cv_data[:3]: # tampilkan 3 contoh

        print("="*40)

        print(f"File: {data['filename']} ({data['role']})")

        print(data['text'][:500], "...") # tampilkan 500 karakter pertama

if __name__ == "__main__":

    test_pdf_extraction()

```

3.1.11 seeding.py

```

import mysql.connector

from faker import Faker

import random

```

```

import os

# ----- CONFIGURATION -----

RESUME_DIRECTORY = "../data/data" # Root folder where roles and CVs are stored

TOTAL_CANDIDATES = 200

DB_CONFIG = {

    'host': "localhost",

    'user': "root",

    'password': "12345",

    'database': "cv_ats_db"

}

# ----- DATABASE SETUP -----

def establish_connection():

    return mysql.connector.connect(**DB_CONFIG)

def setup_database_tables():

    database = establish_connection()

    cursor = database.cursor()

    cursor.execute("""

    CREATE TABLE IF NOT EXISTS ApplicantProfile (

```

```

        applicant_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,

        first_name VARCHAR(50) DEFAULT NULL,

        last_name VARCHAR(50) DEFAULT NULL,

        date_of_birth DATE DEFAULT NULL,

        address VARCHAR(255) DEFAULT NULL,

        phone_number VARCHAR(20) DEFAULT NULL

    );

    """)

    cursor.execute("""

    CREATE TABLE IF NOT EXISTS ApplicationDetail (

        detail_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,

        applicant_id INT NOT NULL,

        application_role VARCHAR(100) DEFAULT NULL,

        cv_path TEXT,

        FOREIGN KEY (applicant_id) REFERENCES ApplicantProfile(applicant_id)

        ON DELETE CASCADE

    );

    """)

    database.commit()

    cursor.close()

    database.close()

    print("Tables created successfully.")

```

```

# ----- LOAD CVS STRUCTURE -----

def fetch_resume_files():

    position_to_resumes = {}

    for position in os.listdir(RESUME_DIRECTORY):

        position_directory = os.path.join(RESUME_DIRECTORY, position)

        if os.path.isdir(position_directory):

            pdf_files = [

                os.path.join(position_directory, file)

                for file in os.listdir(position_directory)

                if file.lower().endswith('.pdf')

            ]

            if pdf_files:

                position_to_resumes[position] = pdf_files

    return position_to_resumes

# ----- FAKE DATA INSERTION -----

def populate_sample_data(candidate_count=TOTAL_CANDIDATES):

    data_generator = Faker()

    database = establish_connection()

    cursor = database.cursor()

    # Load real CV paths from directory

    position_to_resumes = fetch_resume_files()

```

```

if not position_to_resumes:

    print("No valid CVs found in data/ directory.")

    return

complete_resume_list = []

for job_position, file_paths in position_to_resumes.items():

    # Only include .pdf files (case-insensitive)

    pdf_file_paths = [file_path for file_path in file_paths if
file_path.lower().endswith('.pdf')]

    if pdf_file_paths:

        position_to_resumes[job_position] = pdf_file_paths

        for file_path in pdf_file_paths:

            complete_resume_list.append((job_position, file_path))

    else:

        position_to_resumes[job_position] = []

utilized_resume_paths = set()

for _ in range(candidate_count):

    # Insert applicant

    candidate_first_name = data_generator.first_name()

    candidate_last_name = data_generator.last_name()

    birth_date = data_generator.date_of_birth(minimum_age=18, maximum_age=60)

    home_address = data_generator.address().replace("\n", ", ")

```



```

while True:

    contact_number = data_generator.phone_number()

    if len(contact_number) < 20:

        break

    cursor.execute("""

        INSERT INTO ApplicantProfile (first_name, last_name, date_of_birth, address,
phone_number)

        VALUES (%s, %s, %s, %s, %s)

        """, (candidate_first_name, candidate_last_name, birth_date, home_address,
contact_number))

    current_applicant_id = cursor.lastrowid

    # Assign 1-3 different roles with unique CVs

    available_positions = [position for position in position_to_resumes if
position_to_resumes[position]]

    random.shuffle(available_positions)

    selected_role_count = min(random.randint(1, 3), len(available_positions))

    for job_position in available_positions[:selected_role_count]:

        # Filter unused CVs for this role

        unused_resume_files = [resume_file for resume_file in
position_to_resumes[job_position] if resume_file not in utilized_resume_paths]

        if not unused_resume_files:

            continue

```

```

        selected_resume_path = random.choice(unused_resume_files)

        utilized_resume_paths.add(selected_resume_path)

        # Extract only the filename

        resume_filename = os.path.basename(selected_resume_path)

        cursor.execute("""

        INSERT INTO ApplicationDetail (applicant_id, application_role, cv_path)

        VALUES (%s, %s, %s)

        """, (current_applicant_id, job_position.capitalize().replace("-", " "),
resume_filename))

        database.commit()

        cursor.close()

        database.close()

        print(f"{candidate_count} applicants inserted with valid PDF CVs and roles.")

# ----- MAIN EXECUTION -----

if __name__ == "__main__":

    setup_database_tables()

    populate_sample_data()

```

3.1.12 utils.py

```
import os

def generate_tree(path='.', prefix=''):

    try:

        entries = sorted(os.listdir(path))

    except PermissionError:

        return # Skip directories we can't access

    # Filter out unwanted directories/files

    ignored_items = ['venv', 'data', '__pycache__', '.git', 'node_modules']

    entries = [e for e in entries if e not in ignored_items]

    for index, entry in enumerate(entries):

        full_path = os.path.join(path, entry)

        connector = '└─ ' if index == len(entries) - 1 else '│─ '

        print(prefix + connector + entry)

        # Only recurse into directories that are not in ignored list

        if os.path.isdir(full_path) and entry not in ignored_items:

            extension = '    ' if index == len(entries) - 1 else '│  '

            generate_tree(full_path, prefix + extension)

if __name__ == '__main__':

    print(".")
```

```
generate_tree()
```

3.1.13 main.py

```
import flet as ft

import time

import sys

import os

import webbrowser


# Add the parent directory (project/) to Python path

# sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from algo.kmp import kmp_search

from algo.bm import boyer_moore_search

from algo.ahocor import AhoCorasick

from algo.levenshtein import levenshtein_distance, fuzzy_text_search, tune_threshold

from utils.pdf_to_text import load_all_cv_texts

from utils.db import get_applicant_by_cv_filename

from regex.extract_exp import extract_experience_section

from regex.extract_exp import extract_text_from_pdf

from regex.extract_edu import extract_education_section

from regex.extract_skill import extract_skills_from_resume

import re

from utils.seeding import setup_database_tables, populate_sample_data
```

```

# Use heapq for more efficient top-N selection

import heapq

# Dummy data sesuai SQL schema (ApplicantProfile dan ApplicationDetail)

print("📄 Loading CVs from data/data ...")

DUMMY_DATA = load_all_cv_texts("../data/data") # atau "../data/data" tergantung run
location

print(f"✅ Loaded {len(DUMMY_DATA)} CVs.")


# UI Flet untuk CV Analyzer App

def on_view_cv(path: str):

    abs_path = os.path.abspath(path)

    if os.path.exists(abs_path):

        webbrowser.open(f"file://{abs_path}")

    else:

        print(f"❌ File not found: {abs_path}")


def main(page: ft.Page):

    page.title = "CV Analyzer App"

    page.padding = 20

    # Input Keywords

    keywords_field = ft.TextField(

        hint_text="Enter keywords, e.g. React, Express, HTML",

```

```

        width=600,

        border_radius=ft.border_radius.all(20)

    )

    # Pilihan Algoritma Exact Match (KMP/BM)

    algo_dropdown = ft.Dropdown(

        label="Search Algorithm",

        width=200,

        border_radius=ft.border_radius.all(20),

        options=[

            ft.dropdown.Option("KMP"),

            ft.dropdown.Option("Boyer-Moore"),

            ft.dropdown.Option("Aho-Corasick")

        ],

        value="KMP" # nilai default agar selalu ada pilihan

    )

    # Top Matches dropdown

    top_matches = ft.Dropdown(

        label="Top Matches",

        width=100,

        border_radius=ft.border_radius.all(20),

        options=[ft.dropdown.Option(str(i)) for i in range(1, 102)],

        value="3"

    )

```

```

# Fuzzy Matches button (initially hidden)

fuzzy_matches_button = ft.ElevatedButton(

    text="Fuzzy Matches",

    visible=False,

    on_click=lambda e: show_fuzzy_matches_popup()

)

# Store fuzzy match results for popup

fuzzy_match_results = {}

def clear_fuzzy_results():

    nonlocal fuzzy_match_results

    fuzzy_match_results = {}

# Tombol Search

search_button = ft.ElevatedButton(

    text="Search",

    width=600-200-100-12,

    on_click=lambda e: on_search(e),

)

# Header dan container hasil

results_header = ft.Text("Results", size=18, weight=ft.FontWeight.BOLD)

scan_info = ft.Text("0 CVs scanned in 0ms", italic=True)

```

```

results_container = ft.Column(spacing=15)

def on_search(e):

    # Mulai hitung waktu exact match

    t0_exact = time.time()

    keywords = [kw.strip().lower() for kw in (keywords_field.value or '').split(',')]
    if kw.strip():

        exact_matches = []

        # Pre-lowercase all CV texts once

        if not hasattr(DUMMY_DATA[0], '_lower_text'):

            for data in DUMMY_DATA:

                data['_lower_text'] = data['text'].lower()

        # PHASE 1: Exact match search

        search_func = kmp_search if algo_dropdown.value == "KMP" else boyer_moore_search

        for data in DUMMY_DATA:

            total_matches = 0

            details = []

            for kw in keywords:

                if algo_dropdown.value == "Aho-Corasick":

                    positions = AhoCorasick.search(data['text'].lower(), [kw.lower()])

                else:

                    positions = search_func(data['_lower_text'], kw.lower())

```



```

        count = len(positions)

        if count:

            total_matches += count

            details.append((kw, count))

    if total_matches:

        exact_matches.append((data, total_matches, details))

# Sort exact matches by score

exact_matches = heapq.nlargest(len(exact_matches), exact_matches, key=lambda x:
x[1])

top_n = int(top_matches.value or "3")

exact_ms = int((time.time() - t0_exact) * 1000)

# DECISION POINT: Do we need fuzzy search?

if len(exact_matches) >= top_n:

    # We have enough exact matches, no need for fuzzy search

    fuzzy_used = False

    fuzzy_ms = None

    # Just add match_type to exact matches

    final_matches = []

    for data, score, details in exact_matches[:top_n]:

        final_matches.append((data, score, details, "exact"))

```

```

        clear_fuzzy_results()

    else:

        # We need fuzzy search to fill remaining slots

        fuzzy_used = True

        fuzzy_start = time.time()

        clear_fuzzy_results()

        # Create lookup for exact matches for efficiency

        exact_matches_lookup = {id(match_data): (match_score, match_details)

                                for match_data, match_score, match_details in
exact_matches}

        combined_matches = []

        # PHASE 2: Combined exact + fuzzy search

        for data in DUMMY_DATA:

            # Get exact match details if any

            if id(data) in exact_matches_lookup:

                exact_score, exact_details = exact_matches_lookup[id(data)]

                exact_details = list(exact_details)

                exact_keywords_in_cv = {exact_kw for exact_kw, _ in exact_details}

            else:

                exact_score = 0

```

```

        exact_details = []

        exact_keywords_in_cv = set()

        # Run fuzzy search only for keywords without exact matches

        fuzzy_score = 0

        fuzzy_details = []

        for kw in keywords:

            if kw not in exact_keywords_in_cv: # Only fuzzy search if no exact
match
                count, matched_words = fuzzy_text_search(data['text'], kw)

                if count > 0:

                    fuzzy_score += count

                    fuzzy_details.append((kw, count))

                    # Store fuzzy match results for popup

                    if kw not in fuzzy_match_results:

                        fuzzy_match_results[kw] = set()

                        fuzzy_match_results[kw].update(matched_words)

        # Skip if no matches found

        if exact_score == 0 and fuzzy_score == 0:

            continue

        # Determine match type

        if exact_score > 0 and fuzzy_score == 0:

```

```

        match_type = "exact"

    elif exact_score == 0 and fuzzy_score > 0:

        match_type = "fuzzy"

    else:

        match_type = "mixed"

# Combine results

total_display_score = exact_score + fuzzy_score

all_details = exact_details + fuzzy_details

# Priority: exact_score * 1000 + fuzzy_score (exact matches first)

priority_score = exact_score * 1000 + fuzzy_score

    combined_matches.append((data, total_display_score, all_details,
priority_score, match_type))

# Convert fuzzy match results sets to lists for popup display

for kw in fuzzy_match_results:

    if isinstance(fuzzy_match_results[kw], set):

        fuzzy_match_results[kw] = list(fuzzy_match_results[kw])

# Sort by priority score and take top N

combined_matches = heapq.nlargest(len(combined_matches), combined_matches,
key=lambda x: x[3])

    final_matches = [(data, score, details, match_type)

        for data, score, details, _, match_type in
combined_matches[:top_n]]

```

```

        fuzzy_ms = int((time.time() - fuzzy_start) * 1000)

    # Update scan info

    total_found = len(final_matches)

    scan_info.value = f"Exact Match: {len(DUMMY_DATA)} CVs scanned in {exact_ms}ms\n"

    if fuzzy_used:

        scan_info.value += f"Fuzzy Match: {fuzzy_ms}ms\n"

        scan_info.value += f"Showing top {min(top_n, total_found)} of {total_found}
matches"

    # Set matches for pagination

    matches = final_matches

    # Pagination variables

    items_per_page = 5

    current_page = 1

    total_pages = (len(matches) + items_per_page - 1) // items_per_page if matches
else 1

def update_results_display():

    # Update UI

    results_container.controls.clear()

    start_idx = (current_page - 1) * items_per_page

    end_idx = start_idx + items_per_page

    current_matches = matches[start_idx:end_idx]

```

```

# Add pagination controls below cards

if total_pages > 1:

    pagination_controls = ft.Row([

        ft.ElevatedButton(

            text="Previous",

            disabled=current_page == 1,

            on_click=lambda e: change_page(-1)

        ),

        ft.Text(f"Page {current_page} of {total_pages}"),

        ft.ElevatedButton(

            text="Next",

            disabled=current_page == total_pages,

            on_click=lambda e: change_page(1)

        )

    ], alignment=ft.MainAxisAlignment.CENTER, spacing=20)

    results_container.controls.append(pagination_controls)

# Create row for cards only

cards_row = ft.Row(spacing=15,
vertical_alignment=ft.CrossAxisAlignment.START)

for i, (data, total, details, match_type) in enumerate(current_matches,
start_idx + 1):

    # ... existing card creation code ...

```

```

        filename = data.get('filename', 'Unknown')

        lines = [

            ft.Row([

                ft.Container(

                    content=ft.Text(f"#{i}", weight=ft.FontWeight.BOLD,
color="white"),

                    bgcolor="blue",

                    padding=5,

                    border_radius=15,

                    width=50,

                    height=30,

                    alignment=ft.alignment.center

                ),

                ft.Text(filename, weight=ft.FontWeight.BOLD, size=16),

                ft.Container(

                    content=ft.Text(match_type.upper(), size=10, color="white",
weight=ft.FontWeight.BOLD),

                    bgcolor="gray",

                    padding=3,

                    border_radius=8

                )

            ], spacing=10),

            ft.Text(f"{round(total, 2)} match score" if fuzzy_used else
f"{int(total)} matches", italic=True),

            ft.Text("Matched keywords:"),

        ] + [

```

```

        ft.Text(f"• {kw}: {int(score)} match{'es' if score>1 else ''}" if
fuzzy_used else f"• {kw}: {int(score)} occurrence{'s' if score>1 else ''}")

        for kw, score in details

    ] + [

        ft.PopupMenuButton(

            items=[

                ft.PopupMenuItem(

                    text="Summary",

                    on_click=lambda e, name=filename:
show_summary_popup(page, name)

                ),

                ft.PopupMenuItem(

                    text="View CV",

                    on_click=lambda e, path=data['path']: on_view_cv(path)

                ),

                *(

                    [ft.PopupMenuItem(

                        text="Fuzzy Match",

                        on_click=lambda e, card_data=data:
show_fuzzy_matches_popup(card_data),

                    )] if fuzzy_used else []

                ),

            ],

            icon=ft.Icons.MORE_VERT,

            tooltip="Actions"

        )

```



```

    ]

    # Different colors based on match type

    if match_type == "exact":

        bgcolor = "green" if i == 1 else "#006400" if i <= 3 else "#228B22"
# Green shades for exact

        elif match_type == "fuzzy":

            bgcolor = "#FF6347" if i == 1 else "#DC143C" if i <= 3 else "#B22222"
# Red shades for fuzzy

        else: # mixed

            bgcolor = "#FFD700" if i == 1 else "#FFA500" if i <= 3 else "#FF8C00"
# Orange shades for mixed

        cards_row.controls.append(

            ft.Container(

                content=ft.Column(lines),

                padding=10,

                bgcolor=bgcolor,

                width=230,

                border_radius=10

            )

        )

    results_container.controls.append(cards_row)

    page.update()

def change_page(direction):

    nonlocal current_page

    current_page += direction

```

```

        current_page = max(1, min(current_page, total_pages))

        update_results_display()

        # Show/hide fuzzy matches button

        fuzzy_matches_button.visible = fuzzy_used

        page.update()

        # Initial display

        update_results_display()

    # Add filename index at startup

    filename_index = {cv["filename"]: cv for cv in DUMMY_DATA}

    def show_summary_popup(page, filename):

        data = get_applicant_by_cv_filename(filename)

        if data is None:

            dialog = ft.AlertDialog(title=ft.Text("Profile not found"))

        else:

            # Use index for O(1) lookup instead of O(n) search

            cv_data = filename_index.get(filename)

            full_cv_path = cv_data["path"] if cv_data else None

            if full_cv_path:

                print("if path exists:", os.path.exists(full_cv_path))

                print(f"Full CV Path: {full_cv_path}")

            # Create content list starting with basic info

```

```

content_items = [

    ft.Text(f"Name: {data['first_name']} {data['last_name']}"),

    ft.Text(f"Role: {data['application_role']}"),

    ft.Text(f>Date of Birth: {data['date_of_birth']}"),

    ft.Text(f"Address: {data['address']}"),

    ft.Text(f"Phone: {data['phone_number']}"),

    ft.Divider(),

    ft.Text("Work Experience:", weight=ft.FontWeight.BOLD),

]

# Jika file ditemukan, ekstrak pengalaman

# Jika file ditemukan, ekstrak pengalaman dan education

if full_cv_path and os.path.exists(full_cv_path):

    text = extract_text_from_pdf(full_cv_path)

    experiences = extract_experience_section(text)

    education = extract_education_section(text)

    skills = extract_skills_from_resume(text)

    # print("text:", text)

    # print("experiences:", experiences)

    # print("education:", education)

    # print("skills:", skills)

    # Add experience entries

    if experiences:

        for i, exp in enumerate(experiences, 1):

            content_items.append(

                ft.Container(

```

```

        content=ft.Text(f"{i}. {exp}", size=12),

        padding=ft.padding.only(left=10, bottom=5),

        bgcolor="grey",

        border_radius=5

    )

)

else:

    content_items.append(ft.Text("No work experience found",
italic=True))

# Add education section

content_items.extend([

    ft.Divider(),

    ft.Text("Education:", weight=ft.FontWeight.BOLD),

])

if education:

    content_items.append(

        ft.Container(

            content=ft.Text(education, size=12),

            padding=ft.padding.only(left=10, bottom=5),

            bgcolor="lightblue",

            border_radius=5

        )

    )

```

```

        else:
            content_items.append(ft.Text("No education information found",
italic=True))

    # Add skills section

    content_items.extend([

        ft.Divider(),

        ft.Text("Skills:", weight=ft.FontWeight.BOLD),

    ])

    if skills:

        content_items.append(

            ft.Container(

                content=ft.Text(skills, size=12),

                padding=ft.padding.only(left=10, bottom=5),

                bgcolor="lightgreen",

                border_radius=5

            )

        )

    else:

        content_items.append(ft.Text("No skills information found",
italic=True))

    dialog = ft.AlertDialog(

        title=ft.Text("Applicant Summary"),

        content=ft.Column(

```

```

        content_items,

        height=400,

        scroll=ft.ScrollMode.AUTO

    ),

    on_dismiss=lambda e: print("Dialog closed")

)

page.overlay.append(dialog)

dialog.open = True

page.update()

def show_fuzzy_matches_popup(card_data=None):

    if not fuzzy_match_results:

        return

    # Get fuzzy matches specific to this card

    card_fuzzy_matches = {}

    if card_data:

        for kw in fuzzy_match_results.keys():

            count, matched_words = fuzzy_text_search(card_data['text'], kw)

            if count > 0:

                card_fuzzy_matches[kw] = list(matched_words)

    else:

        card_fuzzy_matches = fuzzy_match_results

    content_items = [ft.Text("Fuzzy Match Results:", weight=ft.FontWeight.BOLD,

```

```

size=16)]

    for keyword, typos in card_fuzzy_matches.items():

        content_items.extend([

            ft.Divider(),

            ft.Text(f"Keyword: '{keyword}'", weight=ft.FontWeight.BOLD),

            ft.Text("Similar words found:", italic=True)

        ])

    typos_list = list(typos) if isinstance(typos, set) else typos

    for typo in typos_list[:10]: # Show max 10 typos per keyword

        content_items.append(

            ft.Container(

                content=ft.Text(f"• {typo}", size=12),

                padding=ft.padding.only(left=20, bottom=2),

                bgcolor="lightyellow",

                border_radius=3

            )

        )

    dialog = ft.AlertDialog(

        title=ft.Text("Fuzzy Matches Found"),

        content=ft.Column(

            content_items,

            height=400,

```

```

        scroll=ft.ScrollMode.AUTO

    ),

    on_dismiss=lambda e: print("Fuzzy matches dialog closed")

)

page.overlay.append(dialog)

dialog.open = True

page.update()


# Susun layout

# Susun layout dalam scroll container

main_content = ft.Column([

    ft.Text("CV Analyzer App", size=24, weight=ft.FontWeight.BOLD),

    ft.Row([

        ft.Column([keywords_field]),

        ft.Column([algo_dropdown]),

        ft.Column([top_matches]),

        ft.Column([search_button]),

        ft.Column([fuzzy_matches_button])

    ], alignment=ft.MainAxisAlignment.START, spacing=10),

    ft.Divider(),

    ft.Column([

```



```

        results_header,

        scan_info,

        results_container

    ], spacing=10)

], spacing=20)

# Bungkus dalam Container dengan scroll

scrollable_container = ft.Column(

    controls=[main_content],

    expand=True,

    scroll=ft.ScrollMode.AUTO

)

page.add(scrollable_container)

if __name__ == "__main__":

    # # 1) Setup DB dan isi data sampel MySQL

    # setup_database_tables()

    # populate_sample_data()

    # 2) Jalankan aplikasi Flet

    ft.app(target=main)

```

3.1.14 Tuning Pemilihan Parameter Default

Dalam fungsi `fuzzy_text_search(text: str, target: str, similarity_threshold: float = 0.125, max_distance: int = 4)` -> `tuple[int, list[str]]`, nilai default yang digunakan ditetapkan dengan tujuan utama untuk menjaga agar hasil pencarian tetap akurat, serta meminimalisir kemungkinan kemunculan hasil yang tidak sesuai atau menyesatkan (false positive).

1. `max_distance = 4`

Batas maksimal perubahan ini mengacu pada logika bahwa jika diperlukan lebih dari 4 perubahan (baik penghapusan, penambahan, maupun penggantian huruf), maka kemungkinan besar kata tersebut sudah sangat berbeda dari yang dimaksud. Batas ini penting untuk menjaga keakuratan pencarian, terutama untuk kata atau frasa yang lebih panjang.

2. `similarity_threshold 12.5%`

Ambang toleransi ini dipilih untuk membatasi ruang pencarian fuzzy hanya pada kesalahan ketik kecil yang paling umum terjadi — seperti huruf yang tertukar atau salah ketik sederhana. Dengan cara ini, sistem secara sadar diatur agar tidak menganggap kata yang sudah sangat berbeda makna sebagai cocok, meskipun ada beberapa typo kompleks yang mungkin jadi tidak terdeteksi.

Contoh penerapannya:

- Untuk kata dengan panjang 1–6 huruf, hanya diizinkan maksimal **1** kesalahan.
- Untuk kata sepanjang 7–13 huruf, toleransi naik menjadi **2** kesalahan.
- Kata yang lebih panjang dari 13 huruf bisa mengalami maksimal **3** kesalahan

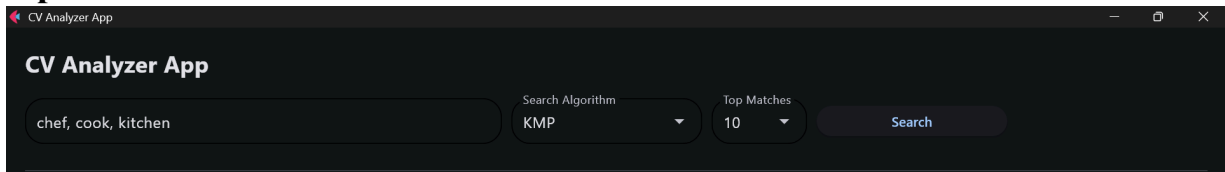
Setelah melalui serangkaian percobaan dan pengujian, kombinasi pengaturan ini terbukti menghasilkan hasil yang paling masuk akal dan sesuai konteks. Jika batas Levenshtein dibuat terlalu longgar, hasilnya justru cenderung mengarah ke kata-kata yang tidak relevan atau menyimpang jauh dari maksud sebenarnya.

3.2 Pengujian

3.2.1 Pengujian KMP

Pengujian 1 (Yang di-input Exact word semua)

Input:



CV Analyzer App

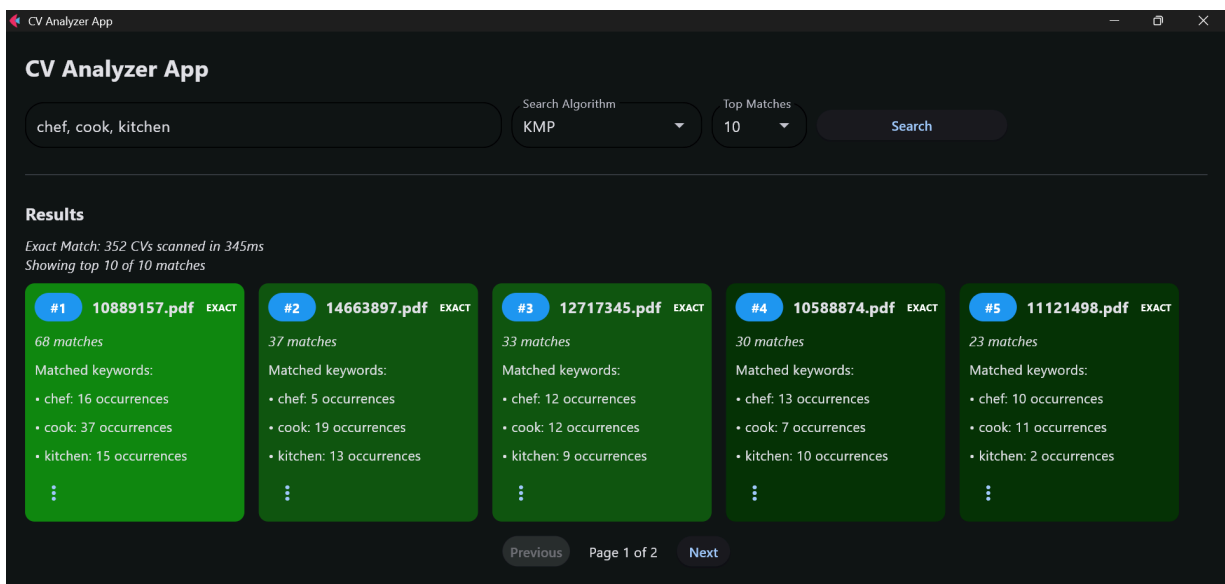
chef, cook, kitchen

Search Algorithm: KMP

Top Matches: 10

Search

Output:



CV Analyzer App

chef, cook, kitchen

Search Algorithm: KMP

Top Matches: 10

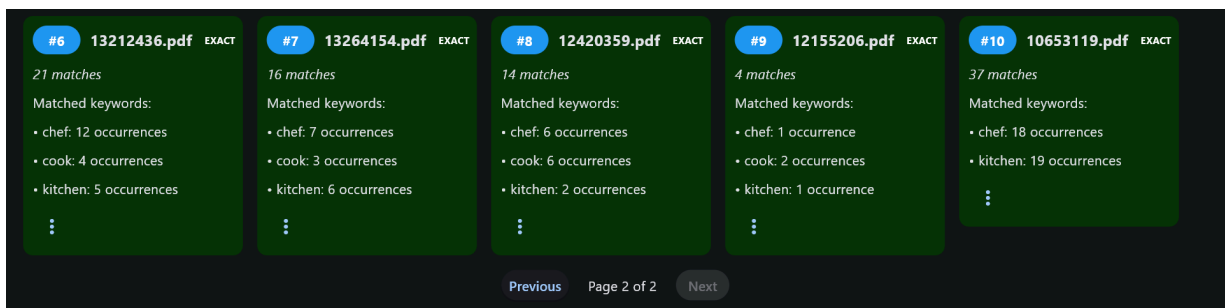
Search

Results

Exact Match: 352 CVs scanned in 345ms
Showing top 10 of 10 matches

#	CV ID	Filename	Match Count	Matched keywords:
#1	10889157.pdf	EXACT	68 matches	<ul style="list-style-type: none">• chef: 16 occurrences• cook: 37 occurrences• kitchen: 15 occurrences
#2	14663897.pdf	EXACT	37 matches	<ul style="list-style-type: none">• chef: 5 occurrences• cook: 19 occurrences• kitchen: 13 occurrences
#3	12717345.pdf	EXACT	33 matches	<ul style="list-style-type: none">• chef: 12 occurrences• cook: 12 occurrences• kitchen: 9 occurrences
#4	10588874.pdf	EXACT	30 matches	<ul style="list-style-type: none">• chef: 13 occurrences• cook: 7 occurrences• kitchen: 10 occurrences
#5	11121498.pdf	EXACT	23 matches	<ul style="list-style-type: none">• chef: 10 occurrences• cook: 11 occurrences• kitchen: 2 occurrences

Previous Page 1 of 2 Next

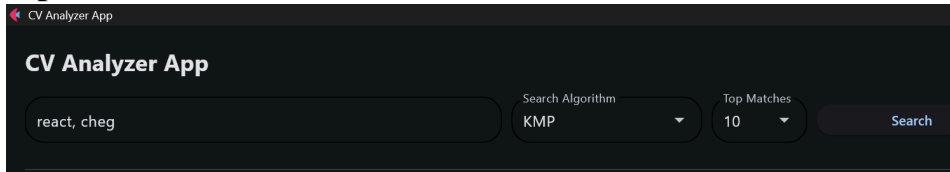


#	CV ID	Filename	Match Count	Matched keywords:
#6	13212436.pdf	EXACT	21 matches	<ul style="list-style-type: none">• chef: 12 occurrences• cook: 4 occurrences• kitchen: 5 occurrences
#7	13264154.pdf	EXACT	16 matches	<ul style="list-style-type: none">• chef: 7 occurrences• cook: 3 occurrences• kitchen: 6 occurrences
#8	12420359.pdf	EXACT	14 matches	<ul style="list-style-type: none">• chef: 6 occurrences• cook: 6 occurrences• kitchen: 2 occurrences
#9	12155206.pdf	EXACT	4 matches	<ul style="list-style-type: none">• chef: 1 occurrence• cook: 2 occurrences• kitchen: 1 occurrence
#10	10653119.pdf	EXACT	37 matches	<ul style="list-style-type: none">• chef: 18 occurrences• kitchen: 19 occurrences

Previous Page 2 of 2 Next

Pengujian 2 (1 Match, 1 Typo)

Input:



CV Analyzer App

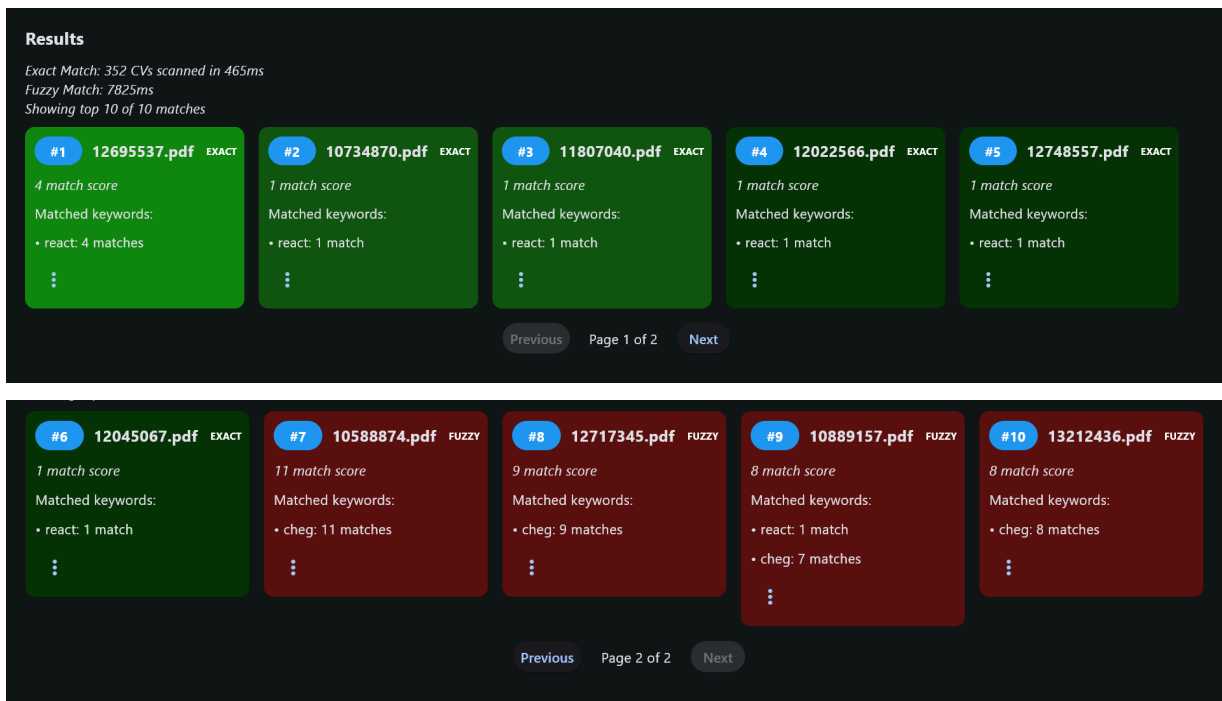
react, cheg

Search Algorithm: KMP

Top Matches: 10

Search

Output:



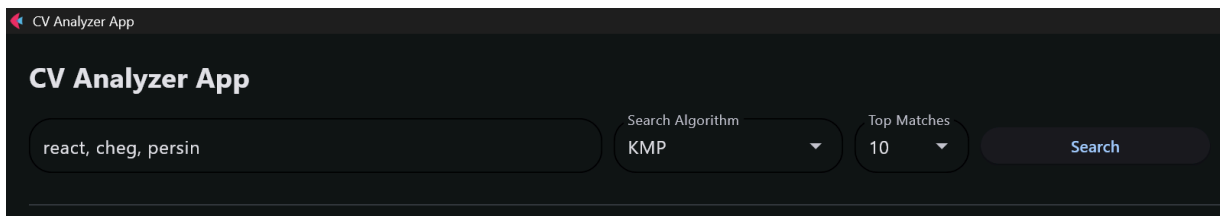
Results

Exact Match: 352 CVs scanned in 465ms
Fuzzy Match: 7825ms
Showing top 10 of 10 matches

#	CV ID	Type	Match Score	Matched keywords
#1	12695537.pdf	EXACT	4 match score	react: 4 matches
#2	10734870.pdf	EXACT	1 match score	react: 1 match
#3	11807040.pdf	EXACT	1 match score	react: 1 match
#4	12022566.pdf	EXACT	1 match score	react: 1 match
#5	12748557.pdf	EXACT	1 match score	react: 1 match
#6	12045067.pdf	EXACT	1 match score	react: 1 match
#7	10588874.pdf	FUZZY	11 match score	cheg: 11 matches
#8	12717345.pdf	FUZZY	9 match score	cheg: 9 matches
#9	10889157.pdf	FUZZY	8 match score	react: 1 match, cheg: 7 matches
#10	13212436.pdf	FUZZY	8 match score	cheg: 8 matches

Pengujian 3 (1 Match, 2 Typo)

Input:



CV Analyzer App

react, cheg, persin

Search Algorithm: KMP

Top Matches: 10

Search

Output:

Results

Exact Match: 352 CVs scanned in 416ms
Fuzzy Match: 6683ms
Showing top 10 of 10 matches

#1 12695537.pdf EXACT
4 match score
Matched keywords:
• react: 4 matches
⋮

#2 10734870.pdf MIXED
2 match score
Matched keywords:
• react: 1 match
• persin: 1 match
⋮

#3 12045067.pdf MIXED
2 match score
Matched keywords:
• react: 1 match
• persin: 1 match
⋮

#4 11807040.pdf EXACT
1 match score
Matched keywords:
• react: 1 match
⋮

#5 12022566.pdf EXACT
1 match score
Matched keywords:
• react: 1 match
⋮

Previous Page 1 of 2 Next

#6 12748557.pdf EXACT
1 match score
Matched keywords:
• react: 1 match
⋮

#7 10889157.pdf FUZZY
8 match score
Matched keywords:
• react: 1 match
• cheg: 7 matches
⋮

#8 11121498.pdf FUZZY
7 match score
Matched keywords:
• cheg: 6 matches
• persin: 1 match
⋮

#9 13411858.pdf FUZZY
6 match score
Matched keywords:
• cheg: 4 matches
• persin: 2 matches
⋮

#10 12155206.pdf FUZZY
2 match score
Matched keywords:
• cheg: 1 match
• persin: 1 match
⋮

Previous Page 2 of 2 Next

3.2.2 Pengujian BM

Pengujian 1 (Yang di-input Exact word semua)

Input:

chef, cook, kitchen

Search Algorithm: Boyer-Moore

Top Matches: 10

Search

Output:

Results

Exact Match: 352 CVs scanned in 289ms
Showing top 10 of 10 matches

#1 10889157.pdf EXACT
68 matches
Matched keywords:
• chef: 16 occurrences
• cook: 37 occurrences
• kitchen: 15 occurrences
⋮

#2 14663897.pdf EXACT
37 matches
Matched keywords:
• chef: 5 occurrences
• cook: 19 occurrences
• kitchen: 13 occurrences
⋮

#3 12717345.pdf EXACT
33 matches
Matched keywords:
• chef: 12 occurrences
• cook: 12 occurrences
• kitchen: 9 occurrences
⋮

#4 10588874.pdf EXACT
30 matches
Matched keywords:
• chef: 13 occurrences
• cook: 7 occurrences
• kitchen: 10 occurrences
⋮

#5 11121498.pdf EXACT
23 matches
Matched keywords:
• chef: 10 occurrences
• cook: 11 occurrences
• kitchen: 2 occurrences
⋮

Previous Page 1 of 2 Next

#6 13212436.pdf EXACT

21 matches

Matched keywords:

- chef: 12 occurrences
- cook: 4 occurrences
- kitchen: 5 occurrences

#7 13264154.pdf EXACT

16 matches

Matched keywords:

- chef: 7 occurrences
- cook: 3 occurrences
- kitchen: 6 occurrences

#8 12420359.pdf EXACT

14 matches

Matched keywords:

- chef: 6 occurrences
- cook: 6 occurrences
- kitchen: 2 occurrences

#9 12155206.pdf EXACT

4 matches

Matched keywords:

- chef: 1 occurrence
- cook: 2 occurrences
- kitchen: 1 occurrence

#10 10653119.pdf EXACT

37 matches

Matched keywords:

- chef: 18 occurrences
- kitchen: 19 occurrences

Previous

Page 2 of 2

Next

Pengujian 2 (1 Match, 1 Typo)

Input:

react, cheg

Search Algorithm

Boyer-Moore

Top Matches

10

Search

Output:

Results

Exact Match: 352 CVs scanned in 223ms

Fuzzy Match: 4791ms

Showing top 10 of 10 matches

#1 12695537.pdf EXACT

4 match score

Matched keywords:

- react: 4 matches

#2 10734870.pdf EXACT

1 match score

Matched keywords:

- react: 1 match

#3 11807040.pdf EXACT

1 match score

Matched keywords:

- react: 1 match

#4 12022566.pdf EXACT

1 match score

Matched keywords:

- react: 1 match

#5 12748557.pdf EXACT

1 match score

Matched keywords:

- react: 1 match

#6 12045067.pdf EXACT

1 match score

Matched keywords:

- react: 1 match

#7 10889157.pdf FUZZY

8 match score

Matched keywords:

- react: 1 match
- cheg: 7 matches

#8 10588874.pdf FUZZY

11 match score

Matched keywords:

- cheg: 11 matches

#9 12717345.pdf FUZZY

9 match score

Matched keywords:

- cheg: 9 matches

#10 13212436.pdf FUZZY

8 match score

Matched keywords:

- cheg: 8 matches

Previous

Page 2 of 2

Next

Pengujian 3 (1 Match, 2 Typo)

Input:

react, cheg, persin

Search Algorithm

Boyer-Moore

Top Matches

10

Search

Output:

Results

Exact Match: 352 CVs scanned in 552ms
 Fuzzy Match: 10764ms
 Showing top 10 of 10 matches

#	CV ID	Type	Match Score	Matched keywords
#1	12695537.pdf	EXACT	4 match score	• react: 4 matches
#2	10734870.pdf	MIXED	2 match score	• react: 1 match • persin: 1 match
#3	12045067.pdf	MIXED	2 match score	• react: 1 match • persin: 1 match
#4	11807040.pdf	EXACT	1 match score	• react: 1 match
#5	12022566.pdf	EXACT	1 match score	• react: 1 match
#6	12748557.pdf	EXACT	1 match score	• react: 1 match
#7	10889157.pdf	FUZZY	8 match score	• react: 1 match • cheg: 7 matches
#8	11121498.pdf	FUZZY	7 match score	• cheg: 6 matches • persin: 1 match
#9	13411858.pdf	FUZZY	6 match score	• cheg: 4 matches • persin: 2 matches
#10	12155206.pdf	FUZZY	2 match score	• cheg: 1 match • persin: 1 match

Previous Page 1 of 2 Next

Previous Page 2 of 2 Next

3.2.3 Pengujian Aho-Corasick

Pengujian 1 (Yang di-input Exact word semua)

Input:

Search Algorithm

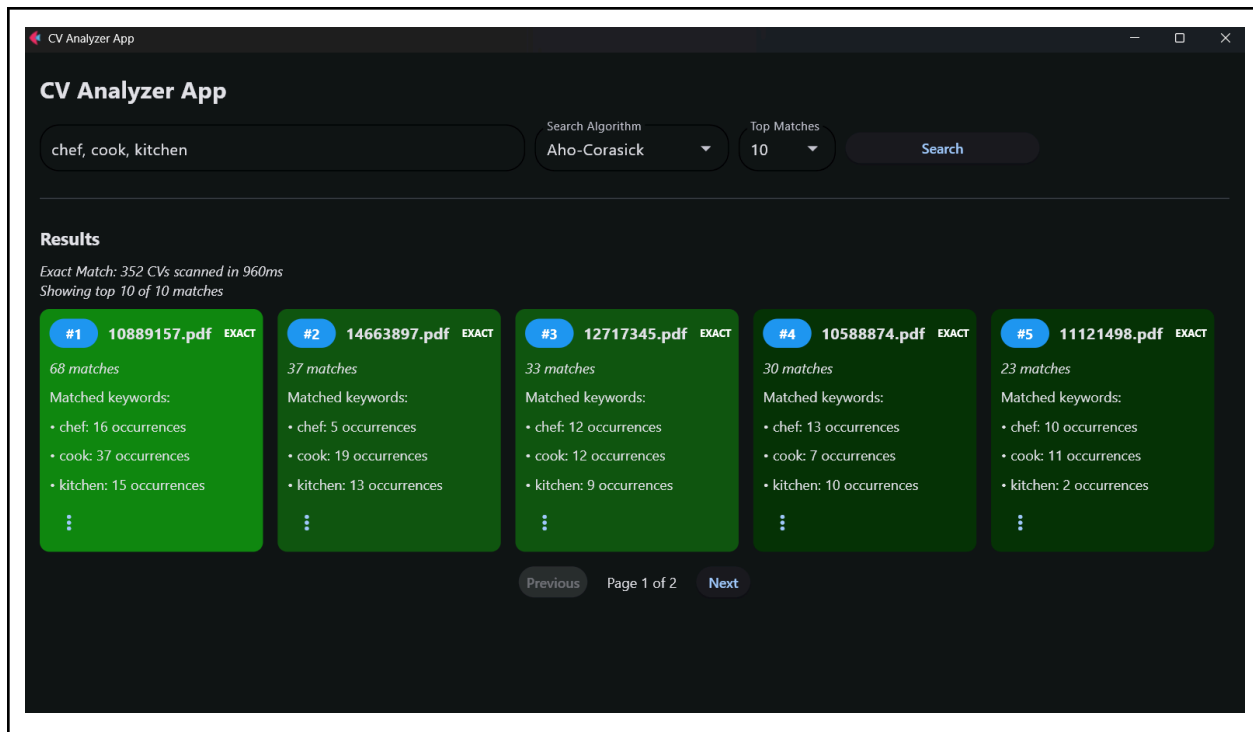
Aho-Corasick

Top Matches

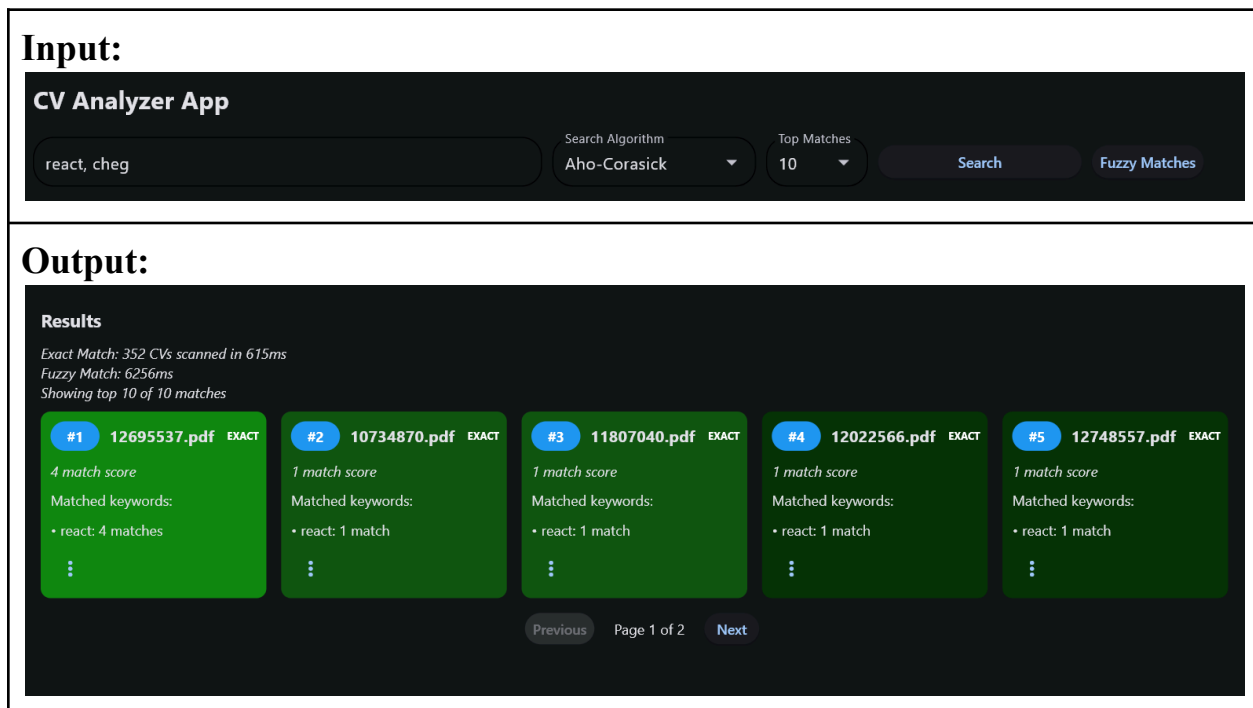
10

Search

Output:



Pengujian 2 (1 Match, 1 Typo)



Results
 Exact Match: 352 CVs scanned in 615ms
 Fuzzy Match: 6256ms
 Showing top 10 of 10 matches

#6	12045067.pdf	EXACT
1 match score		
Matched keywords:		
• react: 1 match		
⋮		

#7	10889157.pdf	FUZZY
8 match score		
Matched keywords:		
• react: 1 match		
• cheg: 7 matches		
⋮		

#8	10588874.pdf	FUZZY
11 match score		
Matched keywords:		
• cheg: 11 matches		
⋮		

#9	12717345.pdf	FUZZY
9 match score		
Matched keywords:		
• cheg: 9 matches		
⋮		

#10	13212436.pdf	FUZZY
8 match score		
Matched keywords:		
• cheg: 8 matches		
⋮		

Previous Page 2 of 2 Next

Pengujian 3 (1 Match, 2 Typo)

Input:

react, cheg, persin

Search Algorithm: Aho-Corasick

Top Matches: 10

Search Fuzzy Matches

Output:

Results
 Exact Match: 352 CVs scanned in 893ms
 Fuzzy Match: 9999ms
 Showing top 10 of 10 matches

#1	12695537.pdf	EXACT
4 match score		
Matched keywords:		
• react: 4 matches		
⋮		

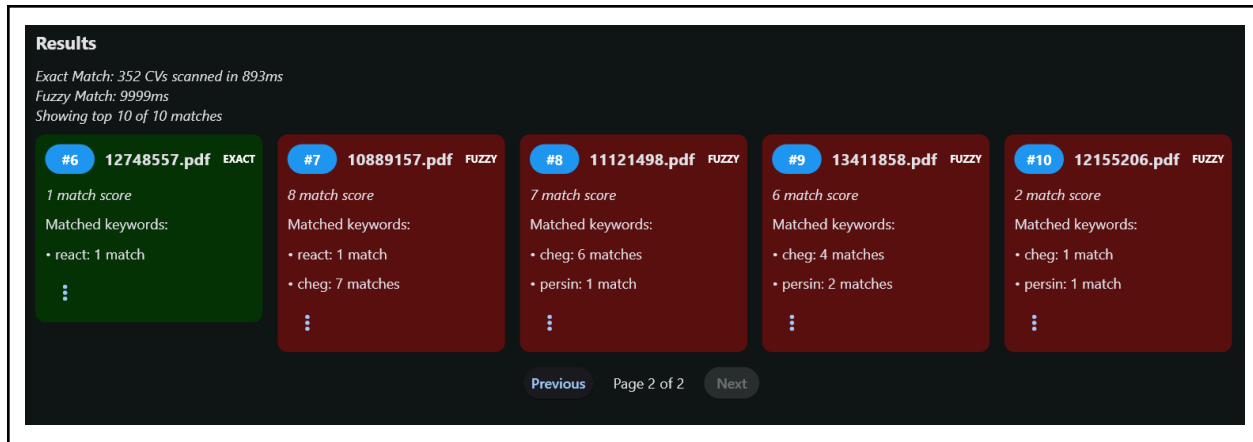
#2	10734870.pdf	MIXED
2 match score		
Matched keywords:		
• react: 1 match		
• persin: 1 match		
⋮		

#3	12045067.pdf	MIXED
2 match score		
Matched keywords:		
• react: 1 match		
• persin: 1 match		
⋮		

#4	11807040.pdf	EXACT
1 match score		
Matched keywords:		
• react: 1 match		
⋮		

#5	12022566.pdf	EXACT
1 match score		
Matched keywords:		
• react: 1 match		
⋮		

Previous Page 1 of 2 Next



3.3 Analisis Hasil Pengujian

Berdasarkan pengujian kami, algoritma yang memiliki rata-rata waktu pencarian yang paling cepat adalah algoritma Boyer-Moore, diikuti dengan Knuth-Morris-Pratt, dan Aho-Corasick. Hal tersebut dikarenakan algoritma Boyer-Moore memang bisa bekerja lebih optimal ketika variasi karakter banyak seperti alfabet, dibanding Knuth-Morris-Pratt yang bekerja lebih baik dengan variasi sedikit seperti rantai DNA. Aho-Corasick menjadi yang paling pelan disini karena harus membuat struktur data *trie* terlebih dahulu dan mencari *failure links* berdasarkan *keywords* yang diberikan, sehingga pada jumlah *keywords* yang sedikit atau teks yang kurang panjang proses tersebut dapat mengalahkan keuntungan waktu pencarian yang diberikan.

Berdasarkan Efekti

BAB 5

Kesimpulan, Saran, dan Refleksi

5.1 Kesimpulan

Algoritma pencocokan string dapat dimanfaatkan secara nyata dalam pembangunan sistem Applicant Tracking System (ATS) berbasis CV digital. Implementasi algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick telah memungkinkan pencarian kata kunci dalam dokumen CV menjadi

lebih cepat, efisien, dan akurat. Sistem ini juga dilengkapi dengan algoritma Levenshtein Distance untuk mendukung pencocokan fuzzy, yang sangat berguna dalam menangani kesalahan penulisan (typo) atau variasi ejaan keyword.

Melalui ekstraksi teks dari PDF, penerapan pattern matching, serta visualisasi hasil pencarian dalam antarmuka grafis yang ramah pengguna, sistem mampu menyaring pelamar secara otomatis berdasarkan kecocokan informasi yang relevan. Dengan demikian, aplikasi ini dapat membantu HR dan *recruiter* dalam mempercepat proses seleksi awal kandidat tanpa harus membaca CV satu per satu secara manual.

5.2 Saran

Untuk pengembangan lebih lanjut, sistem dapat ditingkatkan dengan menambahkan fitur:

- Rekomendasi otomatis berdasarkan *keyword* dan profil kandidat
- Visualisasi statistik hasil pencarian, seperti grafik frekuensi keyword yang paling sering muncul.

Penting juga untuk menguji sistem dengan CV dalam berbagai bahasa dan format, agar ketahanannya terhadap dokumen tidak terstruktur dapat lebih optimal.

5.3 Refleksi

Melalui pengerjaan Tugas Besar ini, kami mendapatkan pemahaman yang lebih mendalam mengenai penerapan algoritma string matching dalam konteks aplikasi nyata. Kami belajar bahwa efisiensi algoritma sangat penting ketika berhadapan dengan data dalam jumlah besar, seperti ratusan dokumen CV. Selain itu, proses mengintegrasikan berbagai komponen, mulai dari ekstraksi PDF, pattern matching, penggunaan regex, hingga tampilan antarmuka pengguna mengajarkan kami bagaimana membangun sistem yang modular dan dapat dipelihara.

Lampiran

Github Repository : https://github.com/Farhanabd05/Tubes3_infokan.git

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	✓	
2	Aplikasi menggunakan basis data berbasis SQL dan berjalan dengan lancar.	✓	

3	Aplikasi dapat mengekstrak informasi penting menggunakan Regular Expression (Regex).	✓	
4	Algoritma <i>Knuth-Morris-Pratt (KMP)</i> dan <i>Boyer-Moore (BM)</i> dapat menemukan kata kunci dengan benar.	✓	
5	Algoritma Levenshtein Distance dapat mengukur kemiripan kata kunci dengan benar.	✓	
6	Aplikasi dapat menampilkan <i>summary CV applicant</i> .	✓	
7	Aplikasi dapat menampilkan <i>CV applicant</i> secara keseluruhan.	✓	
8	Membuat laporan sesuai dengan spesifikasi.	✓	
9	Membuat bonus enkripsi data profil <i>applicant</i> .		✓
10	Membuat bonus algoritma Aho-Corasick.	✓	
11	Membuat bonus video dan diunggah pada Youtube.		✓

Daftar Pustaka

Cao, M. (2010). *Time Complexity of Knuth–Morris–Pratt String Matching Algorithm*. Tufts University.

Gusfield, D. (2009). *Boyer–Moore Algorithm* [PDF lecture notes]. University of California, Davis.

Sukumaran, B. (2023). *The essence of Aho-Corasick algorithm*. Medium.

Haldar, R., & Mukhopadhyay, D. (2011). *Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach*. arXiv.

Friedl, J. E. F. (2006). *Mastering Regular Expressions* (3rd ed.). O'Reilly.

Regex: A complete guide (n.d.). Regex Tutorial.

<https://www.geeksforgeeks.org/dsa/kmp-algorithm-for-pattern-searching/>

<https://www.geeksforgeeks.org/dsa/boyer-moore-algorithm-for-pattern-searching/>