

Tugas Besar 3

IF2211 Strategi Algoritma

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



Disusun oleh:

Ivant Samuel Silaban (13523129)
Ahmad Wafi Idzharulhaqq (13523131)
Rafa Abdussalam Danadyaksa (13523133)

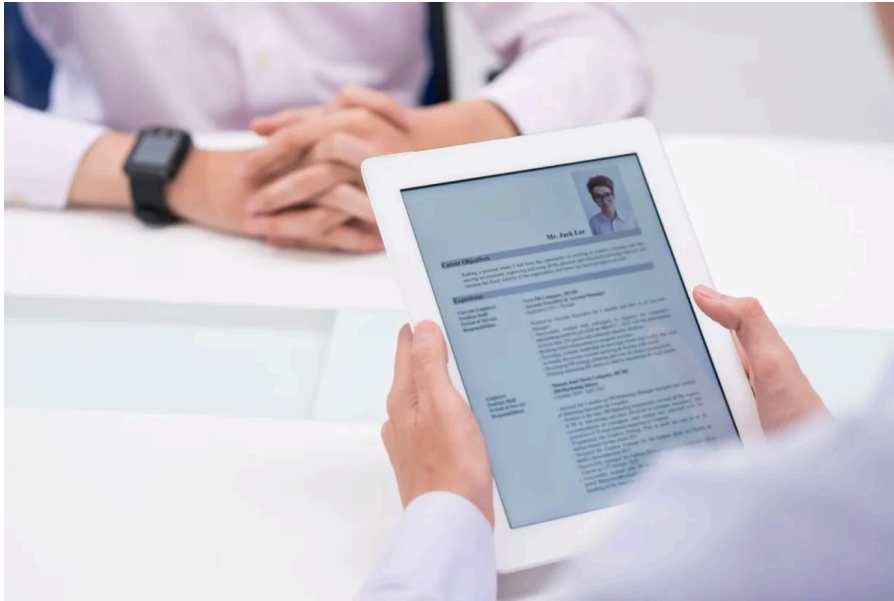
Program Studi Teknik Informatika
Sekolah Teknik Elektro Dan Informatika
Institut Teknologi Bandung
Jl. Ganesa 10, Bandung 40132
2025

DAFTAR ISI

DAFTAR ISI	2
BAB I	3
BAB II	11
2.1 Algoritma Knuth-Morris-Pratt (KMP)	11
2.2 Algoritma Boyer-Moore	11
2.3 Algoritma Aho-Corasick	12
2.4 Regular Expression (Regex)	13
2.5 Levenshtein Distance	13
BAB III	15
3.1. Langkah Pemecahan Masalah	15
3.2. Pemetaan Masalah	15
3.3. Aplikasi Desktop	15
3.4. Fitur Fungsional	16
BAB IV	16
4.1. Implementasi db	16
4.2. Implementasi Core	21
4.3. Implementasi UI	28
4.4. Pengujian	38
BAB V	49
5.1. Kesimpulan	49
5.2. Saran	49
5.3. Refleksi	49
LAMPIRAN	50
Link Penting	50
Tabel Checkpoint	50
DAFTAR PUSTAKA	51

BAB I

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.

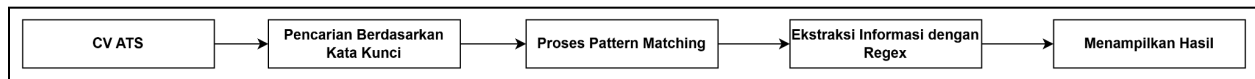
Meskipun demikian, salah satu tantangan besar dalam pengembangan sistem ATS adalah kemampuan untuk memproses dokumen CV dalam format PDF yang tidak selalu terstruktur. Dokumen seperti ini memerlukan metode canggih untuk mengekstrak informasi penting seperti identitas, pengalaman kerja, keahlian, dan 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.

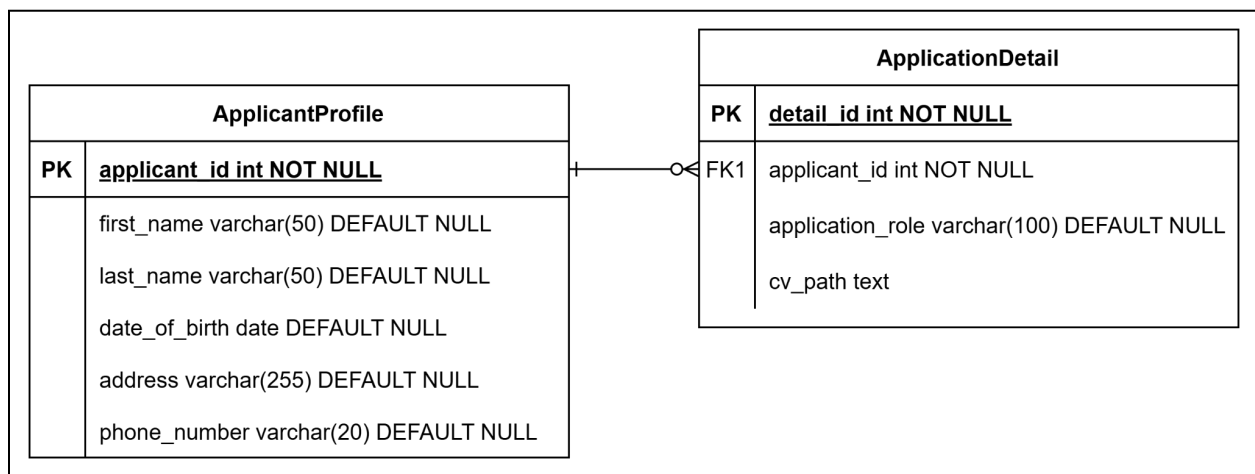
Untuk memberikan pemahaman yang lebih konkret, berikut disajikan contoh kasus penerapan sistem CV ATS beserta prosesnya dan contoh output yang dihasilkan. Dataset yang digunakan dalam contoh ini merupakan dataset CV ATS yang tercantum pada bagian referensi.

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

CV Analyzer App

Keywords :
React, Express, HTML

Search Algorithm:
KMP ☐ BM ☒

Top Matches:
3

Search

Results
100 CVs scanned in 100ms

Candidate	Matches
Farhan	4 matches
Aland	1 match
Ariel	1 match

Farhan (4 matches)
Matched keywords:
1. React: 1 occurrence
2. Express: 2 occurrences
3. HTML: 1 occurrence
[Summary] [View CV]

Aland (1 match)
Matched keywords:
1. React: 1 occurrence
[Summary] [View CV]

Ariel (1 match)
Matched keywords:
1. Express: 1 occurrence
[Summary] [View CV]

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 II

LANDASAN TEORI

2.1 Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah sebuah algoritma pencocokan string yang mencocokkan pola dari kiri ke kanan. Berbeda dengan pendekatan naif (brute force) yang menggeser pattern satu per satu saat terjadi ketidakcocokan, KMP menggunakan sebuah tabel bantu yang disebut Longest Proper Prefix which is also a Suffix (LPS). Proper prefix adalah semua awalan dari sebuah string, kecuali string itu sendiri. Tabel LPS ini dibuat dalam fase pre-processing dengan menganalisis pattern.

Cara kerja KMP melibatkan dua fase utama:

1. Fase Pre-processing (Pembuatan Array LPS):

LPS adalah singkatan dari Longest Proper Prefix which is also a suffix. Untuk setiap sub-pola $pat[0...i]$, $lps[i]$ menyimpan panjang dari awalan terpanjang dari $pat[0...i]$ yang juga merupakan akhir dari $pat[0...i]$. Tabel ini memungkinkan algoritma untuk “melompat” maju secara cerdas saat terjadi ketidakcocokan, karena ia tahu bagian mana dari pattern yang tidak perlu dibandingkan ulang.

2. Fase Pencarian (Matching)

Dalam menggunakan dua pointer (satu untuk teks, satu untuk pattern) dan tabel LPS yang sudah dibuat, algoritma membandingkan teks dan pattern dari kiri ke kanan. Jika terjadi ketidakcocokan, alih-alih menggeser pattern satu posisi, algoritma menggunakan tabel LPS untuk menentukan posisi berikutnya di pattern untuk melanjutkan perbandingan. Hal ini secara efektif melewati banyak perbandingan yang tidak perlu.

Kompleksitas waktu total dari KMP adalah $O(n+m)$, dimana n adalah panjang teks dan m adalah panjang pattern. Ini karena fase pre-processing membutuhkan waktu $O(m)$ dan fase pencarian membutuhkan waktu $O(n)$. Dan kompleksitas ruang yang dibutuhkan adalah $O(m)$ untuk menyimpan array LPS.

2.2 Algoritma Boyer-Moore

Algoritma Boyer-Moore (BM) adalah sebuah algoritma pencocokan string yang mencocokkan pola dari kanan ke kiri. Ketika terjadi ketidakcocokan, BM dapat menggunakan informasi tersebut untuk menggeser pattern dengan “lompatan” yang sangat jauh, seringkali melewati seluruh segmen teks tanpa perlu diperiksa. Efisiensi ini

dicapai melalui kombinasi dua heuristik yaitu Bad Character Heuristic dan Good Suffix Heuristic

Cara kerja BM

1. Bad Character Heuristic

Heuristik ini digunakan saat terjadi ketidakcocokan. Jika karakter di teks tidak cocok dengan karakter di pattern, pattern akan digeser ke kanan hingga karakter tersebut sejajar dengan kemunculan terakhir di dalam pattern. Jika karakter tersebut tidak ada sama sekali di dalam pattern, maka seluruh pattern bisa digeser melewatinya.

2. Good Suffix Heuristic

Heuristic ini fokus pada bagian pattern yang sudah cocok (disebut good suffix). Jika terjadi mismatch, pattern akan digeser untuk menyejajarkan kemunculan lain dari good suffix tersebut di dalam pattern. Jika tidak ada kemunculan lain, ia akan mencari awalan (prefix) dari pattern yang cocok dengan akhiran dari good suffix dan menyejajarkannya.

3. Proses Pencarian

Algoritma akan menyejajarkan pattern dengan teks, lalu membandingkan dari kanan ke kiri. Jika terjadi mismatch, ia akan menghitung pergeseran yang disarankan oleh kedua heuristik dan mengambil nilai maksimum dari keduanya sebagai jarak lompatan.

Kompleksitas waktu untuk algoritma ini adalah $O(n/m)$ untuk best case, ini terjadi jika lompatan-lompatan besar sering terjadi dan $O(n+m)$ untuk worst case. Kompleksitas ruang adalah $O(k)$, dimana k adalah ukuran dari alfabet yang digunakan (untuk tabel bad character).

2.3 Algoritma Aho-Corasick

Algoritma Aho-Corasick adalah algoritma pencocokan string yang dirancang untuk mencari semua kemunculan dari sekumpulan keyword (pola) terbatas di dalam sebuah teks secara bersamaan. Keunggulan utamanya adalah efisiensi yang sangat tinggi karena ia hanya perlu memproses teks input satu kali (single pass), tidak peduli sebanyak apapun jumlah keyword yang dicari. Algoritma ini menggabungkan struktur data Trie dengan konsep failure links yang terinspirasi dari mekanisme state machine pada algoritma KMP

Algoritma ini terbagi dalam dua fase utama, yaitu:

1. Fase Pembangunan Automaton: membangun mesin pencari dari semua kata kunci
2. Fase Pencarian: Menggunakan mesin pencarian untuk memindai teks input

Kompleksitas waktu untuk algoritma ini adalah $O(n + L + z)$, dengan n adalah panjang teks, L adalah total panjang gabungan sebuah keyword yang dicari, dan z adalah jumlah total kemunculan semua keyword di dalam teks.

2.4 Regular Expression (Regex)

Regular Expression (disingkat Regex) adalah urutan karakter khusus yang mendefinisikan sebuah pola pencarian. Alih-alih mencari teks yang literal, Regex memungkinkan kita untuk mencari string atau kumpulan string yang cocok dengan pola tertentu. Misalnya, mencari semua alamat email, nomor telepon, atau kata yang diawali dengan huruf kapital dalam sebuah dokumen. Di Python, fungsionalitas Regex disediakan oleh modul `re`.

Regex bekerja dengan meng-kompilasi pola yang diberikan menjadi sebuah objek pola. Objek ini kemudian digunakan oleh matching engine yang ditulis dalam bahasa C untuk melakukan pencarian pada teks target. Cara kerjanya terdiri dari beberapa komponen utama:

- Karakter Literal: Karakter biasa seperti ...
- Metacharacter: Karakter khusus contohnya
 - Titik
 - `\d`
 - `\w`
- Fungsi: fungsi...

2.5 Levenshtein Distance

Levenshtein Distance, juga dikenal sebagai Edit Distance, adalah metrik yang digunakan untuk mengukur tingkat perbedaan antara dua buah string. Metrik ini menghitung jumlah minimum operasi edit satu karakter yang diperlukan untuk mengubah string pertama menjadi string kedua.

Algoritma ini bekerja dengan menggunakan pendekatan Dynamic Programming untuk menemukan biaya transportasi minimal. Ada tiga jenis operasi yang dipertimbangkan, yang masing-masing memiliki “biaya” 1:

1. Insertion: Menambah sebuah karakter.
2. Deletion: Menghapus sebuah karakter
3. Substitution: Mengganti satu karakter dengan karakter lain.

Prosesnya melibatkan pembuatan matriks 2D berukuran $(\text{panjang string1} + 1) \times (\text{panjang string2} + 1)$. Setiap sel $[i, j]$ dalam matriks ini akan berisi jarak Levenshtein antara i karakter pertama dari string pertama dan j karakter pertama dari string kedua. Nilai setiap

sel dihitung berdasarkan nilai dari sel-sel tetangganya (kiri, atas, dan diagonal) hingga seluruh matriks terisi. Nilai di sel paling kanan-bawah adalah hasil akhirnya.

Kompleksitas waktu algoritma ini adalah $O(m * n)$ karena algoritma ini harus mengisi seluruh matriks, di mana m dan n adalah panjang dari kedua string yang dibandingkan. Kompleksitas ruang yang dibutuhkan juga adalah $O(m * n)$ untuk menyimpan matriks tersebut..

BAB III

ANALISIS PEMECAHAN MASALAH

3.1. Langkah Pemecahan Masalah

Kami diberikan permasalahan berikut: mengembangkan aplikasi CV Analyzer untuk mempermudah seleksi kandidat berdasarkan kata kunci dari CV dalam format PDF. Dengan permasalahan ini, kami juga menghadapi beberapa kendala.

1. Kendala Frontend
 - a. Menggunakan bahasa Python
 - b. Menerima masukan berupa kata kunci pencarian dan jumlah top matches yang diinginkan.
 - c. Dapat memvisualisasikan hasil pencarian dan ringkasan CV dengan antarmuka yang responsif.
 - d. Menampilkan informasi seperti jumlah kecocokan dan waktu eksekusi.
2. Kendala Backend
 - a. Menggunakan bahasa Python
 - b. Membuat struktur data untuk merepresentasikan informasi CV
 - c. Dapat mengimplementasikan KMP dan BM

3.2. Pemetaan Masalah

Pertama, akan dibahas pemetaan masalah pencocokan kata kunci dalam CV menjadi elemen-elemen algoritmik. Kami menggunakan konsep pencarian teks efisien, sebagaimana dijelaskan dalam landasan teori, yang diterapkan melalui algoritma KMP dan BM.

- **KMP:** Teks CV sebagai input utama, kata kunci sebagai pola, tabel LPS untuk menghindari perbandingan berulang, dan logika pencocokan yang mengurangi backtracking untuk kompleksitas $O(n+m)$.
- **BM:** Sama seperti KMP untuk teks dan pola, ditambah tabel bad character dan good suffix untuk pergeseran optimal, mencapai kompleksitas rata-rata $O(n/m)$.

3.3. Aplikasi Desktop

Aplikasi Tubes3_RiceCooker dibangun menggunakan bahasa pemrograman Python dengan framework PyQt5. Penggunaan Python dan PyQt5 dipilih karena kemudahan pengembangan aplikasi desktop, kompatibilitas dengan pustaka seperti PyMuPDF, dan kemampuan untuk menyediakan antarmuka grafis yang interaktif.

3.4. Fitur Fungsional

Aplikasi memiliki beberapa fitur fungsional untuk menyelesaikan permasalahan :

1. Pengguna dapat memasukkan input berupa kata kunci pencarian (misalnya, "React, HTML").
2. Pengguna dapat memilih algoritma yang digunakan (KMP atau BM) dan jumlah top matches yang diinginkan.
3. Pengguna akan mendapatkan hasil dalam bentuk daftar kandidat dengan jumlah kecocokan, serta ringkasan detail (nama, telepon, email, skills, pengalaman, pendidikan) saat memilih kandidat.
4. Pengguna dapat melakukan navigasi kembali ke halaman pencarian dengan tombol "Back to Search".

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi db

4.1.1. Database Manager

db/database_manager.py

```
import mysql.connector
from mysql.connector import Error

class DatabaseManager:
    def __init__(self, config):
        self.config = config
        self.connection = None
        try:
            conn_init = mysql.connector.connect(
                host=self.config['host'],
                user=self.config['user'],
                password=self.config['password']
            )
            cursor_init = conn_init.cursor()
            cursor_init.execute(f"CREATE DATABASE IF NOT EXISTS {self.config['database']}")
            cursor_init.close()
            conn_init.close()
            print(f"Database '{self.config['database']}' is ready.")

            self.connection = mysql.connector.connect(**self.config)
            if self.connection.is_connected():
                self._create_tables()

        except Error as e:
            print(f"Error inisiasi database: {e}")
```



```

        self.connection = None

    def _create_tables(self):
        if not self.connection:
            return

        cursor = self.connection.cursor()
        try:
            # tabel applicant profile
            cursor.execute("""
            CREATE TABLE IF NOT EXISTS ApplicantProfile (
                applicant_id INT AUTO_INCREMENT PRIMARY KEY,
                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
            )
            """)

            # tabel application detail
            cursor.execute("""
            CREATE TABLE IF NOT EXISTS ApplicationDetail (
                detail_id INT AUTO_INCREMENT PRIMARY KEY,
                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
            )
            """)
            self.connection.commit()
            print("Tabel berhasil dibuat.")
        except Error as e:
            print(f"Error membuat tabel: {e}")
        finally:
            cursor.close()

    def get_connection(self):
        if not self.connection or not self.connection.is_connected():
            try:
                self.connection = mysql.connector.connect(**self.config)
            except Error as e:
                print(f"Gagal connect ke database: {e}")
                return None
            return self.connection

    def close(self):
        if self.connection and self.connection.is_connected():
            self.connection.close()
            print("Database closed.")

```

4.1.2. Operations

db/operations.py

```

def _get_db_manager():
    """Menginisialisasi dan mengembalikan instance tunggal DatabaseManager."""
    global _db_manager_instance
    if _db_manager_instance is None:
        try:
            config = configparser.ConfigParser()
            # Menggunakan path absolut dari root direktori
            root_dir = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
            config_path = os.path.join(root_dir, 'src', 'config.ini')

```

```

        if not config.read(config_path):
            raise FileNotFoundError(f"FATAL: File '{config_path}' tidak ditemukan.")

        db_config = dict(config['database'])
        _db_manager_instance = DatabaseManager(db_config)

    except Exception as e:
        print("--- [ERROR KRITIS SAAT SETUP DATABASE] ---")
        traceback.print_exc()
        return None

    return _db_manager_instance

def fetch_dataset_by_category(categories: list[str], limit_per_category: int = 20):
    """
    FUNGSI BARU: Mengambil dataset dari DB sesuai spesifikasi tugas.
    Mengambil data sejumlah `limit_per_category` dari setiap kategori yang diberikan,
    diurutkan secara leksikografis berdasarkan path CV.
    """
    db_manager = _get_db_manager()
    if not db_manager or not db_manager.get_connection():
        print("[ERROR] Gagal mendapatkan koneksi database saat fetch dataset.")
        return []

    conn = db_manager.get_connection()
    # cursor = conn.cursor(dictionary=True)
    cursor = conn.cursor(buffered=True, dictionary=True)
    full_dataset = []

    print("Mulai mengambil dataset berdasarkan kategori...")
    for category in categories:
        query = """
        SELECT
            p.applicant_id as id,
            p.first_name,
            p.last_name,
            d.cv_path
        FROM ApplicantProfile p
        JOIN ApplicationDetail d ON p.applicant_id = d.applicant_id
        WHERE d.cv_path LIKE %s
        ORDER BY d.cv_path ASC
        LIMIT %s
        """
        try:
            category_pattern = f"%/{category}/%"
            cursor.execute(query, (category_pattern, limit_per_category))
            results = cursor.fetchall()
            full_dataset.extend(results)
            # print(f"Mengambil {len(results)} CV dari kategori: {category}") # Uncomment untuk
debug
        except Exception as e:
            print(f"Error fetching data untuk kategori {category}: {e}")

    if cursor:
        cursor.close()

    for candidate in full_dataset:
        candidate['name'] = f"{candidate.get('first_name', '')} {candidate.get('last_name',
        '')}".strip()

    print(f"Total dataset yang diambil: {len(full_dataset)} CV.")
    return full_dataset

def search_cvs(keywords: list[str], algorithm: str, top_n: int):
    """
    Fungsi utama untuk orkestrasi pencarian, menggabungkan Exact dan Fuzzy Match.
    """
    CATEGORIES = [

```

```

'ACCOUNTANT', 'ADVOCATE', 'AGRICULTURE', 'APPAREL', 'ARTS', 'AUTOMOBILE',
'AVIATION', 'BANKING', 'BPO', 'BUSINESS-DEVELOPMENT', 'CHEF', 'CONSTRUCTION',
'CONSULTANT', 'DESIGNER', 'DIGITAL-MEDIA', 'ENGINEERING', 'FINANCE',
'FITNESS', 'HEALTHCARE', 'HR', 'INFORMATION-TECHNOLOGY', 'PUBLIC-RELATIONS',
'SALES', 'TEACHER'
]

all_candidates = fetch_dataset_by_category(CATEGORIES, limit_per_category=20)

if not all_candidates:
    return {'data': [], 'execution_time_exact': 0, 'execution_time_fuzzy': 0, 'total_scanned':
0}

results = {}
lower_keywords = [kw.strip().lower() for kw in keywords if kw.strip()]
total_scanned = len(all_candidates)
failed_files = []

# Bangun automaton Aho-Corasick sekali saja jika dipilih
ac_automaton = None
if algorithm.upper() == 'AHO-CORASICK':
    ac_automaton = AhoCorasick()
    for keyword in lower_keywords:
        ac_automaton.add_keyword(keyword)
    ac_automaton.build_failure_links()

# --- Tahap 1: Exact Matching ---
start_time_exact = time.time()
for candidate in all_candidates:
    cv_path = candidate['cv_path']
    cv_text = extract_text_for_pattern_matching(cv_path)
    if not cv_text:
        continue

    if algorithm.upper() in ['KMP', 'BM']:
        for keyword in lower_keywords:
            search_func = kmp_search if algorithm.upper() == 'KMP' else bm_search
            indices = search_func(cv_text, keyword)
            if indices:
                # (logika KMP/BM tetap sama)
                candidate_id = candidate['id']
                if candidate_id not in results:
                    results[candidate_id] = { 'id': candidate_id, 'name': candidate['name'],
'cv_path': cv_path, 'matched_keywords': {}, 'match_count': 0 }
                    results[candidate_id]['matched_keywords'][keyword] = len(indices)

            elif algorithm.upper() == 'AHO-CORASICK':
                # Cari semua keyword sekaligus
                found_matches = ac_automaton.search(cv_text)
                if found_matches:
                    candidate_id = candidate['id']
                    if candidate_id not in results:
                        results[candidate_id] = { 'id': candidate_id, 'name': candidate['name'],
'cv_path': cv_path, 'matched_keywords': {}, 'match_count': 0 }

                    # Salin hasil dari Aho-Corasick
                    for keyword, indices in found_matches.items():
                        results[candidate_id]['matched_keywords'][keyword] = len(indices)

for res in results.values():
    res['match_count'] = len(res['matched_keywords'])
exact_match_duration = time.time() - start_time_exact

# --- Tahap 2: Fuzzy Matching (Lengkap) ---
start_time_fuzzy = time.time()
found_keywords_exact = {kw for res in results.values() for kw in res['matched_keywords']}
unmatched_keywords = [kw for kw in lower_keywords if kw not in found_keywords_exact]

```

```

fuzzy_match_duration = 0
if unmatched_keywords:
    THRESHOLD = 2 # Jarak Levenshtein <= 2 dianggap mirip

    for candidate in all_candidates:
        if candidate['cv_path'] in failed_files:
            continue

        try:
            cv_text = extract_text_for_pattern_matching(candidate['cv_path'])
            if not cv_text:
                continue

            cv_words = set(cv_text.split())
            for keyword in unmatched_keywords:
                for word in cv_words:
                    if abs(len(keyword) - len(word)) <= THRESHOLD:
                        dist = levenshtein_distance(keyword, word)
                        if dist <= THRESHOLD and dist > 0:
                            candidate_id = candidate['id']
                            if candidate_id not in results:
                                results[candidate_id] = {
                                    'id': candidate_id, 'name': candidate['name'], 'cv_path':
candidate['cv_path'],
                                    'matched_keywords': {}, 'match_count': 0
                                }

                                fuzzy_key = f"{keyword} (similar: {word})"
                                results[candidate_id]['matched_keywords'][fuzzy_key] =
results[candidate_id]['matched_keywords'].get(fuzzy_key, 0) + 1
                                break
                            except Exception as e:
                                logger.error(f"Error dalam fuzzy matching untuk {candidate['cv_path']}: {str(e)}")
                                continue

            for res in results.values():
                res['match_count'] = len(res['matched_keywords'])

            fuzzy_match_duration = time.time() - start_time_fuzzy

# --- Tahap 3: Finalisasi Hasil ---
sorted_results = sorted(results.values(), key=lambda x: x['match_count'], reverse=True)

if failed_files:
    logger.warning(f"Gagal memproses {len(failed_files)} file: {'', '.join(failed_files)}")

return {
    'data': sorted_results[:top_n],
    'execution_time_exact': exact_match_duration,
    'execution_time_fuzzy': fuzzy_match_duration,
    'total_scanned': total_scanned,
    'failed_files': failed_files
}

# def get_applicant_summary(applicant_id: int, cv_path: str):
#     """
#     Mengambil profil dari DB dan mengekstrak info dari CV untuk halaman ringkasan.
#     Fungsi ini sekarang menerima cv_path secara langsung untuk memastikan konsistensi.
#     """
#     db_manager = _get_db_manager()
#     conn = db_manager.get_connection()
#     cursor = conn.cursor(buffered=True, dictionary=True)

#     # Query sekarang hanya untuk mengambil data profil, bukan path lagi
#     cursor.execute("SELECT * FROM ApplicantProfile WHERE applicant_id = %s", (applicant_id,))
#     profile_data = cursor.fetchone()

```

```

#     if not profile_data:
#         cursor.close()
#         return None

#     # Gunakan cv_path yang diberikan langsung, tidak lagi dari hasil query
#     cv_text_for_regex = extract_text_for_regex(cv_path)
#     if not cv_text_for_regex:
#         print(f"Gagal memproses file untuk regex: {cv_path}")

#     extracted_sections = extract_all_sections(cv_text_for_regex)

#     summary_data = {
#         'profile': profile_data,
#         'summary': extracted_sections.get('summary', 'Tidak ditemukan ringkasan.'),
#         'skills': extracted_sections.get('skills', 'Tidak ditemukan keahlian.'),
#         'experience': extracted_sections.get('experience', 'Tidak ditemukan pengalaman kerja.'),
#         'education': extracted_sections.get('education', 'Tidak ditemukan riwayat pendidikan.')
#     }

#     cursor.close()
#     return summary_data

def get_applicant_summary(applicant_id: int, cv_path: str):
    db_manager = _get_db_manager()
    conn = db_manager.get_connection()
    cursor = conn.cursor(buffered=True, dictionary=True)

    cursor.execute("SELECT * FROM ApplicantProfile WHERE applicant_id = %s", (applicant_id,))
    profile_data = cursor.fetchone()

    if not profile_data:
        cursor.close()
        return None

    cv_text_for_regex = extract_text_for_regex(cv_path)

    # Panggil fungsi ekstraksi utama SATU KALI saja
    extracted_sections = extract_all_sections(cv_text_for_regex)

    # Ambil hasilnya dari dictionary yang sudah jadi
    summary_data = {
        'profile': profile_data,
        'summary': extracted_sections.get('summary'),
        'skills': extracted_sections.get('skills'),
        'experience': extracted_sections.get('experience'),
        'education': extracted_sections.get('education')
    }

    cursor.close()
    return summary_data

```

4.2. Implementasi Core

4.2.1. Regex extractor

Fungsi extract_all_section (core/regex_extractor.py)

Input: text: str
Output: dict

```

def extract_raw_sections(text: str) -> dict:
    """Mengekstrak blok teks mentah untuk setiap seksi berdasarkan keywords."""
    cleaned_text = clean_text(text)
    all_kw_pattern = '|'.join(SECTION_KEYWORDS.values())
    title_pattern = re.compile(fr'^\s*({all_kw_pattern})\b\s*?:?', re.IGNORECASE | re.MULTILINE)

```

```

matches = list(title_pattern.finditer(cleaned_text))
sections = {}
for i, match in enumerate(matches):
    title_text = match.group(1).lower()
    section_name = next((name for name, pattern in SECTION_KEYWORDS.items() if
re.fullmatch(pattern, title_text, re.IGNORECASE)), None)
    if section_name and section_name != 'boundary':
        content_start = match.end()
        content_end = matches[i + 1].start() if i + 1 < len(matches) else len(cleaned_text)
        content_block = cleaned_text[content_start:content_end].strip()
        if section_name not in sections: sections[section_name] = []
        sections[section_name].append(content_block)
return sections

def parse_skills(text_blocks: list) -> list:
    """Parser skills yang menggabungkan baris terpotong dan mem-parsing secara kontekstual."""
    if not text_blocks: return []
    full_text = '\n'.join(text_blocks)
    processed_text = re.sub(r'\n(?:[A-Z]*-)', ' ', full_text)
    all_skills = []
    lines = processed_text.split('\n')
    for line in lines:
        line = line.strip()
        if not line: continue
        if ':' in line:
            all_skills.append(re.sub(r'\s+', ' ', line).strip())
        else:
            sub_skills = re.split(r'\s+', line)
            all_skills.extend([skill.strip() for skill in sub_skills if skill.strip()])
    cleaned_skills = [skill.strip(' .,') for skill in all_skills if len(skill.strip(' .,')) > 1]
    return list(OrderedDict.fromkeys(cleaned_skills))

def parse_education(text: str) -> list:
    """Parser pendidikan yang andal."""
    if not text: return []
    parsed_entries = []
    lines = text.strip().split('\n')
    for line in lines:
        line = line.strip()
        if not line: continue
        year, degree, institution, description = 'N/A', 'N/A', 'N/A', None
        if ':' in line:
            parts = line.split(':', 1)
            degree = parts[0].strip()
            institution_part = parts[1].strip()
            year_match = re.search(r'\b((19|20)\d{2})\b', institution_part)
            if year_match:
                year = year_match.group(0)
                institution = re.sub(r'\s+', ' ', institution_part.replace(year, '')).strip()
            else:
                institution = institution_part
        else:
            degree = "Informasi Tambahan"
            institution = "N/A"
            description = line
            year_match = re.search(r'\b((19|20)\d{2})\b', line)
            if year_match: year = year_match.group(0)
            parsed_entry = {'year': year, 'degree': degree, 'institution': institution}
            if description: parsed_entry['description'] = description
            if (parsed_entry.get('degree') != 'N/A' or parsed_entry.get('institution') != 'N/A') and
            parsed_entry not in parsed_entries:
                parsed_entries.append(parsed_entry)
    return parsed_entries

def parse_experience(text: str) -> list:
    """
    Parser experience universal dengan pendekatan "Jangkar Tanggal".
    """

```

```

    if not text:
        return []

    experiences = []
    current_experience = None

    # Pola tanggal yang fleksibel, bisa menangani "Month YYYY" dan "MM/YYYY"
    date_pattern = re.compile(
r'((?:\d{2}/\d{4}|[A-Za-z]+\s+\d{4})\s*to\s*(?:\d{2}/\d{4}|[A-Za-z]+\s+\d{4}|Present|Current))',
        re.IGNORECASE
    )

    lines = text.strip().split('\n')

    for line in lines:
        line = line.strip()
        if not line:
            continue

        date_match = date_pattern.search(line)

        # Jika sebuah baris mengandung pola tanggal, anggap itu awal entri baru.
        if date_match:
            # Simpan dulu entri sebelumnya jika ada
            if current_experience:
                current_experience['description'] =
'\n'.join(current_experience['description']).strip()
                experiences.append(current_experience)

            # Buat entri baru
            date_range = date_match.group(1).strip()

            # Teks sebelum tanggal adalah Perusahaan
            company = line[:date_match.start()].strip()
            # Teks setelah tanggal adalah Posisi
            position = line[date_match.end():].strip()

            current_experience = {
                'date_range': date_range,
                'company': company.replace('Company Name', '').strip(' '),
                'position': position.strip(' '),
                'description': [] # Siapkan list untuk menampung deskripsi
            }

            # Jika bukan baris tanggal, maka ini adalah bagian dari deskripsi pekerjaan saat ini.
            elif current_experience:
                current_experience['description'].append(line)

        # Simpan entri pekerjaan terakhir setelah loop selesai
        if current_experience:
            current_experience['description'] = '\n'.join(current_experience['description']).strip()
            experiences.append(current_experience)

    return experiences

def extract_all_sections(text: str) -> dict:
    """Fungsi utama untuk mengekstrak semua informasi dari teks CV."""
    sections = extract_raw_sections(text)

    skills_blocks = sections.get('skills', [])
    skills_list = parse_skills(skills_blocks)

    experience_text = '\n'.join(sections.get('experience', []))
    experience_list = parse_experience(experience_text)

    education_text = '\n'.join(sections.get('education', []))
    education_list = parse_education(education_text)

```

```

summary_text = '\n'.join(sections.get('summary', []))

# Mengembalikan dengan kunci 'experience' sesuai permintaan Anda.
return {
    'summary': clean_text(summary_text).replace("\n", " ") or 'N/A',
    'skills': skills_list,
    'experience': experience_list,
    'education': education_list or []
}

```

4.2.2. pdf_parser

Fungsi extract_text_for_regex (core/pdf_parser.py)

Input: pdf_path: str

Output: str

```

def extract_text_for_regex(pdf_path: str) -> str:
    """
    Mengekstrak teks dari PDF dengan mempertahankan format asli (case dan newlines).
    Cocok untuk digunakan dengan Regular Expressions.

    Args:
        pdf_path: Path menuju file PDF.

    Returns:
        String teks dengan format asli.
    """
    try:
        absolute_path = get_absolute_path(pdf_path)
        if not absolute_path:
            return ""

        doc = fitz.open(absolute_path)
        full_text = ""
        for page in doc:
            # Menggunakan get_text("text") adalah default dan mempertahankan format
            full_text += page.get_text("text")
        doc.close()
        return full_text
    except Exception as e:
        logger.error(f"Error reading PDF for Regex {pdf_path}: {str(e)}")
        return ""

def extract_text_for_pattern_matching(pdf_path: str) -> str:
    """
    Mengekstrak teks dari PDF dan mengubahnya menjadi format 'flat' dan lowercase.
    Cocok untuk digunakan dengan algoritma KMP dan Boyer-Moore.

    Args:
        pdf_path: Path menuju file PDF.

    Returns:
        String teks dalam format lowercase dan spasi tunggal.
    """
    try:
        # Kita bisa menggunakan fungsi pertama sebagai dasar
        raw_text = extract_text_for_regex(pdf_path)
        if not raw_text:
            logger.warning(f"Tidak ada teks yang diekstrak dari {pdf_path}")
            return ""
    
```



```

# 1. Ubah ke huruf kecil (lowercase)
text = raw_text.lower()

# 2. Ganti semua karakter whitespace (spasi, tab, newline) dengan satu spasi
text = re.sub(r'\s+', ' ', text)

return text.strip()
except Exception as e:
    logger.error(f"Error dalam extract_text_for_pattern_matching untuk {pdf_path}: {str(e)}")
return ""

```

4.2.3. levenshtein

Fungsi levenshtein_distance (core/levenshtein.py)

Input: s1: str, s2: str
Output: int

```

def levenshtein_distance(s1: str, s2: str) -> int:
    """
    Menghitung Levenshtein distance antara dua string.
    """
    s1, s2 = s1.lower(), s2.lower()
    m, n = len(s1), len(s2)

    # Inisialisasi matriks DP
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    # Isi matriks
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            cost = 0 if s1[i - 1] == s2[j - 1] else 1
            dp[i][j] = min(dp[i - 1][j] + 1,          # Deletion
                           dp[i][j - 1] + 1,        # Insertion
                           dp[i - 1][j - 1] + cost) # Substitution

    return dp[m][n]

```

4.2.4. kmp

Fungsi kmp_search (core/kmp.py)

Input: text: str, pattern: str
Output: list[int]

```

def compute_lps_array(pattern: str) -> list[int]:
    """Menghitung tabel LPS untuk algoritma KMP."""
    length = 0
    lps = [0] * len(pattern)
    i = 1

```

```

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

def kmp_search(text: str, pattern: str) -> list[int]:
    """
    Mencari semua kemunculan pattern dalam text menggunakan KMP.

    Returns:
        List berisi indeks awal dari semua kemunculan pattern.
    """
    if not pattern:
        return []

    m, n = len(pattern), len(text)
    lps = compute_lps_array(pattern)
    indices = []
    i = 0 # Indeks untuk text
    j = 0 # Indeks untuk pattern

    while i < n:
        if pattern[j] == text[i]:
            i += 1
            j += 1

        if j == m:
            indices.append(i - j)
            j = lps[j - 1]
        elif i < n and pattern[j] != text[i]:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1
    return indices

```

4.2.5. Boyer Moore

Fungsi bm_search (core/bm.py)

Input: text: str, pattern: str

Output: list[int]

```

def preprocess_good_suffix(pattern: str) -> list[int]:
    """
    Melakukan pre-processing untuk membuat tabel shift Good Suffix.
    """
    m = len(pattern)
    shift = [0] * (m + 1)
    border_pos = [0] * (m + 1)

    # Inisialisasi
    i = m
    j = m + 1
    border_pos[i] = j

```

```

# Case 1: Mencari good suffix yang muncul lagi di pattern
while i > 0:
    while j <= m and pattern[i - 1] != pattern[j - 1]:
        if shift[j] == 0:
            shift[j] = j - i
            j = border_pos[j]
        i -= 1
        j -= 1
    border_pos[i] = j

# Case 2: Sebagian good suffix cocok dengan prefix pattern
j = border_pos[0]
for i in range(m + 1):
    if shift[i] == 0:
        shift[i] = j
    if i == j:
        j = border_pos[j]

return shift

def bm_search(text: str, pattern: str) -> list[int]:
    """
    Mencari semua kemunculan pattern dalam text menggunakan Boyer-Moore
    dengan Bad Character dan Good Suffix Heuristics.
    """
    if not pattern or not text or len(pattern) > len(text):
        return []

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

    # Pre-processing
    bad_char_table = build_bad_char_table(pattern)
    good_suffix_shift_table = preprocess_good_suffix(pattern)

    indices = []
    s = 0 # s adalah shift dari pattern relatif terhadap text

    while s <= n - m:
        j = m - 1 # Mulai perbandingan dari kanan ke kiri

        while j >= 0 and pattern[j] == text[s + j]:
            j -= 1

        if j < 0:
            # Pattern ditemukan
            indices.append(s)
            # Geser pattern berdasarkan tabel good suffix untuk mencari kemunculan berikutnya
            s += good_suffix_shift_table[0]
        else:
            # Mismatch terjadi
            # Hitung pergeseran dari kedua heuristik
            bad_char_shift = j - bad_char_table.get(text[s + j], -1)
            good_suffix_shift = good_suffix_shift_table[j + 1]

            # Pilih pergeseran terbesar untuk memaksimalkan lompatan
            s += max(bad_char_shift, good_suffix_shift)

    return indices

```

4.2.6. Aho Corasick

```
core/aho_corasick.py
```

```
class AhoCorasick:
```

```

"""Implementasi algoritma Aho-Corasick."""
def __init__(self):
    self.root = TrieNode()

def add_keyword(self, keyword: str):
    """Menambahkan sebuah keyword ke dalam Trie."""
    node = self.root
    for char in keyword:
        node = node.children.setdefault(char, TrieNode())
    node.output.append(keyword)

def build_failure_links(self):
    """Membangun failure links untuk semua node menggunakan Breadth-First Search (BFS)."""
    queue = deque()
    # Inisialisasi failure links untuk anak-anak dari root
    for node in self.root.children.values():
        node.fail = self.root
        queue.append(node)

    while queue:
        current_node = queue.popleft()
        for char, next_node in current_node.children.items():
            queue.append(next_node)
            fail_node = current_node.fail
            # Cari failure link untuk next_node
            while char not in fail_node.children and fail_node is not self.root:
                fail_node = fail_node.fail

            if char in fail_node.children:
                next_node.fail = fail_node.children[char]
            else:
                next_node.fail = self.root

            # Gabungkan output dari failure link
            next_node.output.extend(next_node.fail.output)

def search(self, text: str) -> dict:
    """
    Mencari semua keyword dalam teks dan mengembalikan posisinya.
    Return: dict dalam format {'keyword': [end_index_1, end_index_2, ...]}
    """
    results = {}
    node = self.root
    for i, char in enumerate(text):
        while char not in node.children and node is not self.root:
            node = node.fail

        if char in node.children:
            node = node.children[char]
        else:
            # Jika tidak ada transisi bahkan dari root, tetap di root
            node = self.root

        # Jika ada output di node saat ini, catat semua keyword yang cocok
        if node.output:
            for keyword in node.output:
                if keyword not in results:
                    results[keyword] = []
                # Mencatat posisi akhir dari keyword yang ditemukan
                results[keyword].append(i)

    return results

```

4.3. Implementasi UI

4.3.1. Main Page

ui/main_page.py

```
class CVAnalyzerApp(QMainWindow):
    """Main window for the CV Analyzer application."""
    def __init__(self):
        super().__init__()
        self.setWindowTitle("CV Analyzer App")
        self.setGeometry(100, 100, 800, 600) # Ukuran window lebih besar
        self.setStyleSheet("""
            QMainWindow {
                background-color: #f5f5f5;
            }
            QLabel {
                color: #333333;
            }
            QLineEdit {
                padding: 8px;
                border: 2px solid #e0e0e0;
                border-radius: 6px;
                background-color: white;
            }
            QLineEdit:focus {
                border: 2px solid #4CAF50;
            }
            QRadioButton {
                spacing: 8px;
                color: #333333;
            }
            QRadioButton::indicator {
                width: 18px;
                height: 18px;
            }
            QScrollArea {
                border: none;
                background-color: transparent;
            }
        """)

        # Central widget and stack for switching pages
        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)
        self.stack = QStackedWidget()
        layout = QVBoxLayout(self.central_widget)
        layout.setSpacing(20) # Menambah spacing antar elemen
        layout.setContentsMargins(30, 30, 30, 30) # Menambah margin
        layout.addWidget(self.stack)

        # Create search page (this class) and summary page
        self.search_page = QWidget()
        self.init_search_ui()
        self.summary_page = SummaryWindow(self)
        self.stack.addWidget(self.search_page)
        self.stack.addWidget(self.summary_page)

        # Show search page by default
        self.stack.setCurrentWidget(self.search_page)

    def init_search_ui(self):
        """Initialize the UI for the search page."""
        layout = QVBoxLayout(self.search_page)
        layout.setSpacing(20)
        layout.setContentsMargins(20, 20, 20, 20)

        # Title dengan frame
        title_frame = QFrame()
        title_frame.setStyleSheet("""
            QFrame {

```

```

        background-color: #4CAF50;
        border-radius: 10px;
        padding: 10px;
    }
    """
    title_layout = QVBoxLayout(title_frame)
    title_label = QLabel("CV Analyzer App")
    title_label.setFont(QFont("Segoe UI", 24, QFont.Bold))
    title_label.setStyleSheet("color: white;")
    title_label.setAlignment(Qt.AlignCenter)
    title_layout.addWidget(title_label)
    layout.addWidget(title_frame)

    # Input container dengan frame
    input_frame = QFrame()
    input_frame.setStyleSheet("""
        QFrame {
            background-color: white;
            border-radius: 10px;
            padding: 20px;
        }
    """)
    input_layout = QVBoxLayout(input_frame)

    # Keywords input
    keywords_layout = QHBoxLayout()
    keywords_label = QLabel("Keywords:")
    keywords_label.setFont(QFont("Segoe UI", 11, QFont.Bold))
    self.keywords_input = QLineEdit("React, Express, HTML")
    self.keywords_input.setFont(QFont("Segoe UI", 11))
    self.keywords_input.setPlaceholderText("Masukkan kata kunci (pisahkan dengan koma)")
    keywords_layout.addWidget(keywords_label)
    keywords_layout.addWidget(self.keywords_input)
    input_layout.addLayout(keywords_layout)

    # Algorithm selection
    algo_layout = QHBoxLayout()
    algo_label = QLabel("Search Algorithm:")
    algo_label.setFont(QFont("Segoe UI", 11, QFont.Bold))
    self.kmp_radio = QRadioButton("KMP")
    self.bm_radio = QRadioButton("BM")
    self.ac_radio = QRadioButton("Aho-Corasick")
    self.kmp_radio.setChecked(True)
    for radio in [self.kmp_radio, self.bm_radio, self.ac_radio]:
        radio.setFont(QFont("Segoe UI", 11))
    algo_layout.addWidget(algo_label)
    algo_layout.addWidget(self.kmp_radio)
    algo_layout.addWidget(self.bm_radio)
    algo_layout.addWidget(self.ac_radio)
    algo_layout.addStretch()
    input_layout.addLayout(algo_layout)

    # Top Matches input
    top_matches_layout = QHBoxLayout()
    top_matches_label = QLabel("Top Matches:")
    top_matches_label.setFont(QFont("Segoe UI", 11, QFont.Bold))
    self.top_matches_input = QSpinBox()
    self.top_matches_input.setRange(1, 99)
    self.top_matches_input.setValue(3)
    self.top_matches_input.setFont(QFont("Segoe UI", 11))
    self.top_matches_input.setStyleSheet("""
        QSpinBox {
            padding: 5px;
            border: 2px solid #e0e0e0;
            border-radius: 6px;
            min-width: 80px;
        }
    """)

```

```

top_matches_layout.addWidget(top_matches_label)
top_matches_layout.addWidget(self.top_matches_input)
top_matches_layout.addStretch()
input_layout.addLayout(top_matches_layout)

layout.addWidget(input_frame)

# Search button
self.search_button = QPushButton("Search")
self.search_button.setFont(QFont("Segoe UI", 14, QFont.Bold))
self.search_button.setStyleSheet("""
    QPushButton {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #4CAF50, stop:1 #2E7D32);
        color: white;
        padding: 15px;
        border-radius: 8px;
        min-height: 50px;
    }
    QPushButton:hover {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #66BB6A, stop:1 #4CAF50);
    }
    QPushButton:pressed {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #2E7D32, stop:1 #1B5E20);
    }
""")
self.search_button.clicked.connect(self.perform_search)
layout.addWidget(self.search_button)

# Summary result section
self.result_summary = QLabel("")
self.result_summary.setFont(QFont("Segoe UI", 11))
self.result_summary.setStyleSheet("""
    QLabel {
        background-color: #e8f5e9;
        padding: 10px;
        border-radius: 6px;
        color: #2E7D32;
    }
""")
layout.addWidget(self.result_summary)

# Scroll area for search results
self.scroll_area = QScrollArea()
self.scroll_area.setWidgetResizable(True)
self.scroll_area.setStyleSheet("""
    QScrollArea {
        border: none;
        background-color: transparent;
    }
    QScrollBar:vertical {
        border: none;
        background: #f0f0f0;
        width: 10px;
        margin: 0px;
    }
    QScrollBar::handle:vertical {
        background: #c0c0c0;
        min-height: 20px;
        border-radius: 5px;
    }
    QScrollBar::add-line:vertical, QScrollBar::sub-line:vertical {
        height: 0px;
    }
""")
self.results_container = QWidget()
self.results_layout = QVBoxLayout(self.results_container)
self.results_layout.setSpacing(15)
self.scroll_area.setWidget(self.results_container)

```

```

        layout.addWidget(self.scroll_area)

    def perform_search(self):
        # 1. Kumpulkan input dari UI
        keywords_text = self.keywords_input.text()
        if not keywords_text.strip():
            QMessageBox.warning(self, "Input Kosong", "Silakan masukkan kata kunci.")
            return

        keywords = [kw.strip() for kw in keywords_text.split(',')]
        algorithm = 'KMP' if self.kmp_radio.isChecked() else 'BM' if self.bm_radio.isChecked() else
        'AHO-CORASICK' if self.ac_radio.isChecked() else 'KMP'
        top_n = int(self.top_matches_input.text())

        # 2. Panggil fungsi backend tunggal
        # Seluruh logika kompleks (PDF, KMP/BM, Levenshtein, DB) ada di dalam search_cvs
        search_result = search_cvs(keywords, algorithm, top_n)

        # 3. Tampilkan hasil yang dikembalikan oleh backend
        self.result_summary.setText(
            f"Exact Match: scanned in {search_result['execution_time_exact']:.2f}s.\n"
            f"Fuzzy Match: scanned in {search_result['execution_time_fuzzy']:.2f}s."
        )

        # Hapus hasil pencarian sebelumnya
        while self.results_layout.count():
            child = self.results_layout.takeAt(0)
            if child.widget():
                child.widget().deleteLater()

        # Tampilkan kartu kandidat baru
        if not search_result['data']:
            self.results_layout.addWidget(QLabel("Tidak ada CV yang cocok ditemukan.))
        else:
            for candidate_data in search_result['data']:
                # data `candidate_data` sudah dalam format yang benar dari backend
                card = CandidateCard(candidate_data, self.switch_to_summary, self.view_cv)
                self.results_layout.addWidget(card)

        self.results_layout.addStretch()

    def switch_to_summary(self, candidate_data):
        """Beralih ke halaman ringkasan setelah mengambil data dari backend."""
        applicant_id = candidate_data['id']
        cv_path = candidate_data['cv_path'] # Ambil cv_path dari data kartu

        # Panggil backend dengan DUA argumen untuk memastikan file yang benar diproses
        summary_data = get_applicant_summary(applicant_id, cv_path)

        if summary_data:
            self.summary_page.update_candidate_info(summary_data)
            self.stack.setCurrentWidget(self.summary_page)
        else:
            QMessageBox.critical(self, "Error", f"Tidak dapat menemukan detail untuk applicant ID:
            {applicant_id}")

    def switch_to_search(self):
        """Switch back to the search page."""
        self.stack.setCurrentWidget(self.search_page)

    def view_cv(self, cv_path):
        """Membuka file CV menggunakan penampil default sistem."""
        try:
            # Dapatkan path absolut dari root project
            project_root = os.path.abspath(os.path.join(os.path.dirname(__file__), '..', '..'))
            # Gabungkan dengan path relatif dari database
            absolute_path = os.path.join(project_root, cv_path)

```



```

        if os.path.exists(absolute_path):
            QDesktopServices.openUrl(QUrl.fromLocalFile(absolute_path))
        else:
            QMessageBox.warning(self, "File Tidak Ditemukan",
                                f"File CV tidak dapat ditemukan di:\n{absolute_path}\n\n"
                                f"Path relatif: {cv_path}\n"
                                f"Root project: {project_root}")
    except Exception as e:
        QMessageBox.critical(self, "Error", f"Gagal membuka file CV: {e}")

def closeEvent(self, event):
    """Menutup koneksi database saat aplikasi ditutup."""
    print("Menutup aplikasi dan koneksi database...")
    close_db_connection()
    event.accept()

```

4.3.2. Summary Page

ui/summary_page.py

```

class SummaryWindow(QWidget):
    """Widget untuk menampilkan halaman ringkasan CV yang telah di-parsing."""
    def __init__(self, controller):
        super().__init__()
        self.controller = controller
        self.setStyleSheet("""
            QWidget { background-color: #f5f5f5; }
            QLabel { color: #333333; font-family: "Segoe UI"; }
            QTextEdit { background-color: white; border: 1px solid #e0e0e0; border-radius: 8px;
padding: 10px; }
            QFrame { background-color: white; border-radius: 10px; padding: 20px; }
            QScrollArea { border: none; }
            QScrollBar:vertical { border: none; background: #e0e0e0; width: 10px; margin: 0px; }
            QScrollBar::handle:vertical { background: #b0b0b0; min-height: 20px; border-radius: 5px;
}
        """)
        self.init_ui()

    def init_ui(self):
        """Menginisialisasi semua elemen UI pada halaman."""
        layout = QVBoxLayout(self)
        layout.setSpacing(20)
        layout.setContentsMargins(30, 30, 30, 30)

        title_frame = QFrame()
        title_frame.setStyleSheet("background-color: #4CAF50; border-radius: 10px; padding: 15px;")
        title_layout = QVBoxLayout(title_frame)
        title_label = QLabel("CV Summary")
        title_label.setFont(QFont("Segoe UI", 24, QFont.Bold))
        title_label.setStyleSheet("color: white;")
        title_label.setAlignment(Qt.AlignCenter)
        title_layout.addWidget(title_label)
        layout.addWidget(title_frame)

        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)

        self.info_container = QWidget()
        self.info_layout = QVBoxLayout(self.info_container)
        self.info_layout.setSpacing(15)
        self.info_layout.setAlignment(Qt.AlignTop)
        scroll_area.setWidget(self.info_container)
        layout.addWidget(scroll_area)

        back_button = QPushButton("Back to Search")

```

```

        back_button.setFont(QFont("Segoe UI", 12, QFont.Bold))
        back_button.setMinimumHeight(45)
        back_button.setStyleSheet("""
            QPushButton { background: #616161; color: white; padding: 12px; border-radius: 8px;
border: none; }
            QPushButton:hover { background: #757575; }
            QPushButton:pressed { background: #515151; }
        """)
        back_button.clicked.connect(self.controller.switch_to_search)
        layout.addWidget(back_button)

    def _create_section_header(self, title):
        """Helper untuk membuat judul seksi yang konsisten."""
        header_label = QLabel(title)
        header_label.setFont(QFont("Segoe UI", 14, QFont.Bold))
        header_label.setStyleSheet("color: #333; margin-top: 10px; margin-bottom: 5px;")
        return header_label

    def update_candidate_info(self, summary_data: dict):
        """Memperbarui UI dengan data kandidat yang telah di-parsing."""
        while self.info_layout.count():
            child = self.info_layout.takeAt(0)
            if child.widget():
                child.widget().deleteLater()

        profile = summary_data.get('profile', {})
        name = f"{profile.get('first_name', '')} {profile.get('last_name', '')}".strip()
        birthdate_raw = profile.get('date_of_birth', 'N/A')
        address = profile.get('address', 'N/A')
        phone = profile.get('phone_number', 'N/A')

        birthdate = str(birthdate_raw)
        if isinstance(birthdate_raw, datetime.date):
            birthdate = birthdate_raw.strftime('%d %B %Y')

        skills_list = summary_data.get('skills', [])
        experience_list = summary_data.get('experience', [])
        education_list = summary_data.get('education', [])

        # --- Tampilkan Informasi Pribadi ---
        personal_frame = QFrame()
        personal_layout = QVBoxLayout(personal_frame)
        name_label = QLabel(name)
        name_label.setFont(QFont("Segoe UI", 18, QFont.Bold))
        personal_layout.addWidget(name_label)
        for label, value in [("Birthdate", birthdate), ("Address", address), ("Phone", phone)]:
            info_layout = QHBoxLayout()
            label_widget = QLabel(f"<b>{label}</b>")
            value_widget = QLabel(str(value))
            info_layout.addWidget(label_widget)
            info_layout.addWidget(value_widget, 1)
            personal_layout.addLayout(info_layout)
        self.info_layout.addWidget(personal_frame)

        # --- Tampilkan Keahlian (Skills) - Menggunakan kembali gaya Tombol/Tag ---
        self.info_layout.addWidget(self._create_section_header("Skills"))
        skills_frame = QFrame()
        skills_layout = QVBoxLayout(skills_frame)
        if skills_list and isinstance(skills_list, list):
            current_row = QHBoxLayout()
            current_row.setSpacing(10)
            skills_per_row = 3 # Jumlah skill per baris
            skill_count = 0

            for skill in skills_list:
                if skill.strip():
                    skill_tag = QPushButton(skill.strip())
                    skill_tag.setFont(QFont("Segoe UI", 10))

```

```

        skill_tag.setStyleSheet("""
            QPushButton {
                background-color: #E8F5E9;
                color: #2E7D32;
                border: 1px solid #C8E6C9;
                border-radius: 12px;
                padding: 8px 15px;
            }
        """)
        skill_tag.setSizePolicy(QSizePolicy.Preferred, QSizePolicy.Fixed)
        current_row.addWidget(skill_tag)
        skill_count += 1

        if skill_count % skills_per_row == 0:
            skills_layout.addLayout(current_row)
            current_row = QHBoxLayout()
            current_row.setSpacing(10)

        if current_row.count() > 0:
            current_row.addStretch()
            skills_layout.addLayout(current_row)
        else:
            skills_layout.addWidget(QLabel("Tidak ada data keahlian."))
        self.info_layout.addWidget(skills_frame)

        # --- Tampilkan Pengalaman Kerja ---
        self.info_layout.addWidget(self._create_section_header("Job History"))
        if experience_list:
            for exp in experience_list:
                exp_frame = QFrame()
                exp_frame.setStyleSheet("background-color: #fafafa; border: 1px solid #eee; padding:
15px; border-radius: 8px;")
                exp_layout = QVBoxLayout(exp_frame)

                pos_label = QLabel(exp.get('position', 'N/A'))
                pos_label.setFont(QFont("Segoe UI", 12, QFont.Bold))
                pos_label.setStyleSheet("color: #2E7D32;")

                comp_date_text = f"{exp.get('company', 'N/A')} | {exp.get('date_range', 'N/A')}"
                comp_date_label = QLabel(comp_date_text)
                comp_date_label.setStyleSheet("color: #666; margin-bottom: 10px; font-style:
italic;")

                desc_label = QLabel(exp.get('description', 'Tidak ada deskripsi.').replace('\n',
'<br>'))
                desc_label.setWordWrap(True)

                exp_layout.addWidget(pos_label)
                exp_layout.addWidget(comp_date_label)
                exp_layout.addWidget(desc_label)
                self.info_layout.addWidget(exp_frame)
            else:
                no_exp_frame = QFrame()
                no_exp_frame.setLayout(QVBoxLayout())
                no_exp_frame.layout().addWidget(QLabel("Tidak ada data pengalaman kerja."))
                self.info_layout.addWidget(no_exp_frame)

        # --- Tampilkan Riwayat Pendidikan ---
        self.info_layout.addWidget(self._create_section_header("Education"))
        if education_list:
            for edu in education_list:
                edu_frame = QFrame()
                edu_frame.setStyleSheet("background-color: #fafafa; border: 1px solid #eee; padding:
15px; border-radius: 8px;")
                edu_layout = QVBoxLayout(edu_frame)

                degree_label = QLabel(edu.get('degree', 'N/A'))
                degree_label.setFont(QFont("Segoe UI", 12, QFont.Bold))

```

```

        degree_label.setStyleSheet("color: #2E7D32;")

        inst_year_text = f"{edu.get('institution', 'N/A')} | {edu.get('year', 'N/A')}}"
        inst_year_label = QLabel(inst_year_text)
        inst_year_label.setStyleSheet("color: #666;")

        edu_layout.addWidget(degree_label)
        edu_layout.addWidget(inst_year_label)

        if edu.get('description'):
            desc_label = QLabel(edu.get('description'))
            desc_label.setWordWrap(True)
            desc_label.setStyleSheet("font-style: italic; color: #555;")
            edu_layout.addWidget(desc_label)

        self.info_layout.addWidget(edu_frame)
    else:
        no_edu_frame = QFrame()
        no_edu_frame.setLayout(QVBoxLayout())
        no_edu_frame.layout().addWidget(QLabel("Tidak ada data riwayat pendidikan."))
        self.info_layout.addWidget(no_edu_frame)

    self.info_layout.addStretch()

```

4.3.3. Widgets

ui/widgets.py

```

class CandidateCard(QFrame):
    """Custom widget for displaying a candidate card in search results."""
    def __init__(self, candidate, on_summary_click, on_view_click):
        super().__init__()
        self.candidate = candidate
        self.on_summary_click = on_summary_click
        self.on_view_click = on_view_click
        self.setStyleSheet("""
            QFrame {
                background-color: white;
                border-radius: 10px;
                padding: 15px;
            }
            QLabel {
                color: #333333;
            }
        """)
        self.init_ui()

    def init_ui(self):
        layout = QVBoxLayout(self)
        layout.setSpacing(12)

        # Candidate name dengan style yang lebih baik
        name_label = QLabel(self.candidate.get("name", "N/A"))
        name_label.setFont(QFont("Segoe UI", 14, QFont.Bold))
        name_label.setStyleSheet("color: #2E7D32;")
        layout.addWidget(name_label)

        # Matched keywords dengan style yang lebih baik
        matches = self.candidate.get("match_count", 0)
        matches_label = QLabel(f"Matched keywords: {matches}")
        matches_label.setFont(QFont("Segoe UI", 11))
        matches_label.setStyleSheet("""
            QLabel {
                background-color: #E8F5E9;
                color: #2E7D32;
                padding: 5px 10px;
            }
        """)

```

```

        border-radius: 5px;
    }
    """
    layout.addWidget(matches_label)

# Keywords with occurrences dengan style yang lebih baik
keywords = self.candidate.get("matched_keywords", {})
if keywords:
    keywords_frame = QFrame()
    keywords_frame.setStyleSheet("""
        QFrame {
            background-color: #F5F5F5;
            border-radius: 8px;
            padding: 10px;
        }
    """)
    keywords_layout = QVBoxLayout(keywords_frame)
    keywords_layout.setSpacing(5)

    for keyword, count in keywords.items():
        keyword_layout = QHBoxLayout()
        keyword_label = QLabel(f"• {keyword}")
        keyword_label.setFont(QFont("Segoe UI", 10))
        count_label = QLabel(f"{count} occurrence{'s' if count > 1 else ''}")
        count_label.setFont(QFont("Segoe UI", 10))
        count_label.setStyleSheet("color: #757575;")
        keyword_layout.addWidget(keyword_label)
        keyword_layout.addWidget(count_label)
        keyword_layout.addStretch()
        keywords_layout.addLayout(keyword_layout)

    layout.addWidget(keywords_frame)

# Action buttons dengan style yang lebih baik
button_layout = QHBoxLayout()
button_layout.setSpacing(10)

summary_button = QPushButton("Summary")
summary_button.setFont(QFont("Segoe UI", 11, QFont.Bold))
summary_button.setStyleSheet("""
    QPushButton {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #2196F3, stop:1 #1976D2);
        color: white;
        padding: 10px 20px;
        border-radius: 8px;
        min-width: 120px;
    }
    QPushButton:hover {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #42A5F5, stop:1 #2196F3);
    }
    QPushButton:pressed {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #1976D2, stop:1 #1565C0);
    }
""")
summary_button.clicked.connect(lambda: self.on_summary_click(self.candidate))

view_button = QPushButton("View CV")
view_button.setFont(QFont("Segoe UI", 11, QFont.Bold))
view_button.setStyleSheet("""
    QPushButton {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #FF9800, stop:1 #F57C00);
        color: white;
        padding: 10px 20px;
        border-radius: 8px;
        min-width: 120px;
    }
    QPushButton:hover {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #FFA726, stop:1 #FF9800);
    }
""")

```

```

    }
    QPushButton:pressed {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #F57C00, stop:1 #EF6C00);
    }
    """
    view_button.clicked.connect(lambda: self.on_view_click(self.candidate.get("cv_path", "")))

    button_layout.addWidget(summary_button)
    button_layout.addWidget(view_button)
    layout.addLayout(button_layout)

```

4.4. Pengujian

Test Case 1: (cooking - KMP - 3)

Output:

CV Analyzer App

Keywords:

Search Algorithm: ☒ KMP ☐ BM ☐ Aho-Corasick

Top Matches:

Search

Exact Match: scanned in 7.56s.
Fuzzy Match: scanned in 0.00s.

RaD3n FrAnC1sco

Matched keywords: 1

• cooking 1 occurrence

SummaryView CV

AR13L H3RFR1S0N

Matched keywords: 1

• cooking 2 occurrences

SummaryView CV

Ari3L HerFR150n

Matched keywords: 1

• cooking 1 occurrence

SummaryView CV

CV Summary

RaD3n FrAnC1sco

Birthdate:

19 November 2003

Address:

Jl. Apel No. 13, Lhokseumawe

Phone:

081212343434

Skills

Communication: Speaks effectively, articulate, concise, listens attentively, can think on my feet, feels confident talking to people, persuades others, provides feedbacks, openly expresses ideas

Interpersonal: Motivates others,

Organizational: Punctual, multi-task, meets deadlines, sets goals, manages projects

Computer: Mastery of Microsoft Office Programs (Excel, Word, PowerPoint,

Flexible: Willing to try new things, able to work on schedule, interested in improving efficiency on any task

Calm under pressur

Back to Search

CV Summary

Job History

Administration Office Assistant

Company Name City , State | 06/2010 to 08/2010

(Summer Job) Worked with the Director of the Cultural Affairs department in filing papers, answering phone calls, assisting on historic preservation projects, working with clients, and educating young students about the importance of preserving the island's culture and language.

Administration Office Assistant and Public Safety Trainee

Company Name City , State | 07/2011 to 08/2011

Department of Public Safety (Rota, M.P., 96951, Songsang Village, District 3, CHM)
Assisted on paper works with public safety, arranged meetings, answered phone calls, filed paper works, assisted on traffic, worked with police officers on radar speed detection on highways, patrolled around the island for any vehicles not conforming to the law, had CPR training, worked with fire department on fire safety rules.

Back to Search

CV Summary

Company Name City , State | 02/2013 to 09/2013

Cooperative Education

Program Workforce (Rota High School, N.P., 96951, Sangseng Village, CIMI) Teacher Aide for High School students

Assisted in tutoring students who are below average, worked with teachers on projects and plans to help improve both math and English departments, made assignments to help students practice their skills, made educational games, worked with SPED students, assisted on parent/teacher meetings, and joined hand in hand with teachers and staff to evaluate the progress of students throughout the school year.

Student Activities Office Assistant 08/2013 to 12/2014 Company Name City , State

Helped organized activities in the University's Campus. Worked with other Universities to create combined events. Assisted clubs and organizations for sponsored activities volunteering opportunities

Education

Diploma : General Rota High School City , State

High School

2013

Bachelor of Arts : Elementary Education University of Portland City , State

2017

N/A

Back to Search

Analisis: Program berhasil melakukan pencarian terhadap keyword ‘cooking’ menggunakan algoritma KMP dengan menampilkan 3 hasil teratas. Pencarian berlangsung selama 7.56 detik dan mendapatkan 3 data yang relevan. Dapat terlihat bahwa menu CV summary bekerja dan dapat menampilkan detail summary dengan baik.

Test Case 2: (cooking - BM - 3)

Output:

CV Analyzer App

Keywords:

Search Algorithm: ☐ KMP ☒ BM ☐ Aho-Corasick

Top Matches:

Search

Exact Match: scanned in 5.27s.

Fuzzy Match: scanned in 0.00s.

RaD3n FrAnC1sco

Matched keywords: 1

• cooking 1 occurrence

AR13L H3RFR1S0N

Matched keywords: 1

• cooking 2 occurrences

Summary

View CV

Ari3L HerfR150n

Matched keywords: 1

• cooking 1 occurrence

ADMINISTRATION OFFICE ASSISTANT

Summary

Enthusiastic student-teacher with superb leadership and communication skills. Easily cultivates trusting and productive relationships with students, parents, teachers administration, and others. Effective at providing quality instruction and fostering a positive working environment with excellent interpersonal and organization skills.

Highlights

- Communication: Speaks effectively, articulate, concise, listens attentively, can think on my feet, feels confident talking to people, persuades others, provides feedbacks, openly expresses ideas
- Interpersonal: Motivates others, understands others, works well with others, supportive, cooperative, counsels, and accepts responsibility
- Management: Leads others, makes decisions, takes charge or initiative, can teach or mentor others
- Organizational: Punctual, multi-task, meets deadlines, sets goals, manages projects
- Computer: Mastery of Microsoft Office Programs (Excel, Word, PowerPoint, Outlook), Ability to work with several operating systems
- Attention to Detail: Produces work that is neat and attractive, ensures that tasks are all done
- Flexible: Willing to try new things, able to work on schedule, interested in improving efficiency on any task
- Calm under pressure
- Decisive
- Curriculum development
- Organizational development knowledge
- Member of Portland Helping Hands and Family Homeless Shelter

Accomplishments

- Rota High School, 2009-2013: Class Vice-President, National Honor Society President, Youth Advisory President, Anti-Bullying Campaign President, Take Action Youth Advocacy Member, Junior Achievement Program Public Relations Officer, Army ROTC Company Commander, Won Most Outstanding Female Graduate, and Leadership Award
- Founded and led a comprehensive after school enrichment program at Rota Elementary School: "The Reading Bridge Project"
- University of Portland Student, Class of 2017, Elementary Education Major, Army ROTC, and Kappa Delta Pi (Education) Honor Society Officer
- Led 3 Summer Camp Programs, and student teach in 3 schools at the Portland District

Experience

Administration Office Assistant 06/2010 to 08/2010 Company Name City, State

(Summer Job) Worked with the Director of the Cultural Affairs department in filing papers, answering phone calls, assisting on historic preservation projects, working with clients, and educating young students about the importance of preserving the island's culture and language. 2. Department of Public Safety (Rota, M.P., 96951, Songsong Village, District 3, CNMI)

Administration Office Assistant and Public Safety Trainee 07/2011 to 08/2011 Company Name City, State

Assisted on minor works with public safety, attended meetings, answered phone calls, filed minor works, assisted on traffic, worked with police

CV Summary

RaD3n FrAnC1sco

Birthdate: 19 November 2003

Address: Jl. Apel No. 13, Lhokseumawe

Phone: 081212343434

Skills

Communication: Speaks effectively, articulate, concise, listens attentively, can think on my feet, feels confident talking to people, persuades others, provides feedbacks, openly expresses ideas

Interpersonal: Motivates others,

Organizational: Punctual, multi-task, meets deadlines, sets goals, manages projects

Computer: Mastery of Microsoft Office Programs (Excel, Word, PowerPoint,

Flexible: Willing to try new things, able to work on schedule, interested in improving efficiency on any task

Calm under pressure

Curriculum development

Organizational development I

Back to Search

Analisis: Program berhasil melakukan pencarian terhadap keyword ‘cooking’ menggunakan

algoritma BM dengan menampilkan 3 hasil teratas. Pencarian berlangsung selama 5.27 detik dan mendapatkan 3 data yang relevan. Dapat terlihat bahwa menu CV summary bekerja dan dapat menampilkan detail summary dengan baik. Selain itu, melihat pada waktu yang dibutuhkan bagi pencarian untuk menemukan data, didapati bahwa algoritma BM memiliki waktu pencarian yang lebih singkat dibandingkan KMP. Hal ini dapat mendukung argumen bahwa dalam mayoritas kasus, algoritma BM memiliki sistem yang lebih baik untuk pencarian dibandingkan KMP.

Test Case 3: (accounting - KMP - 2)

Output:

The screenshot displays the CV Analyzer App interface. At the top, a green header bar contains the text "CV Analyzer App". Below this, a search form is visible with the following fields and options:

- Keywords:** A text input field containing the word "accounting".
- Search Algorithm:** Radio buttons for "KMP" (selected), "BM", and "Aho-Corasic".
- Top Matches:** A dropdown menu showing the number "2".

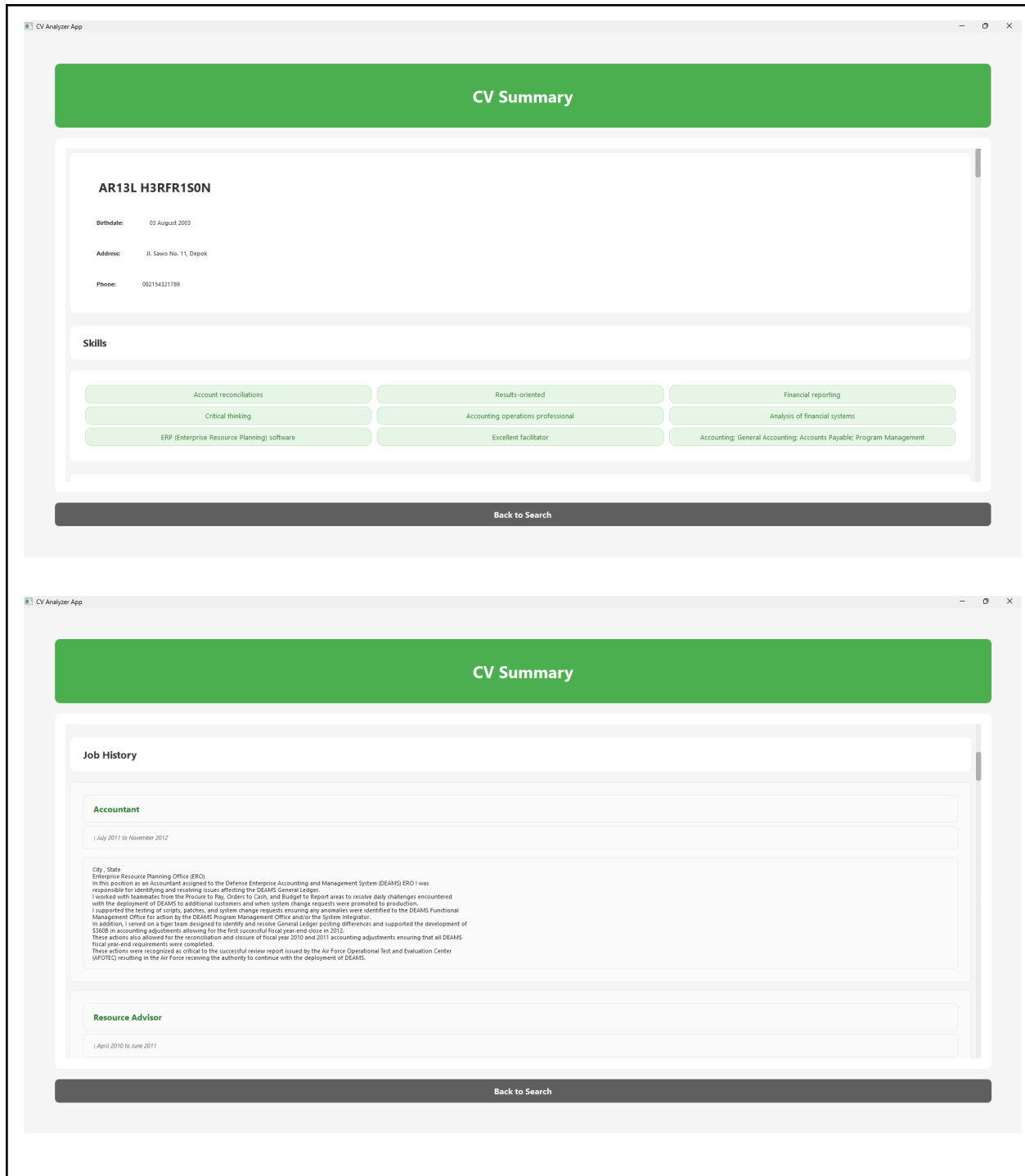
A green "Search" button is located below the search form. Below the button, a green bar displays the search results:

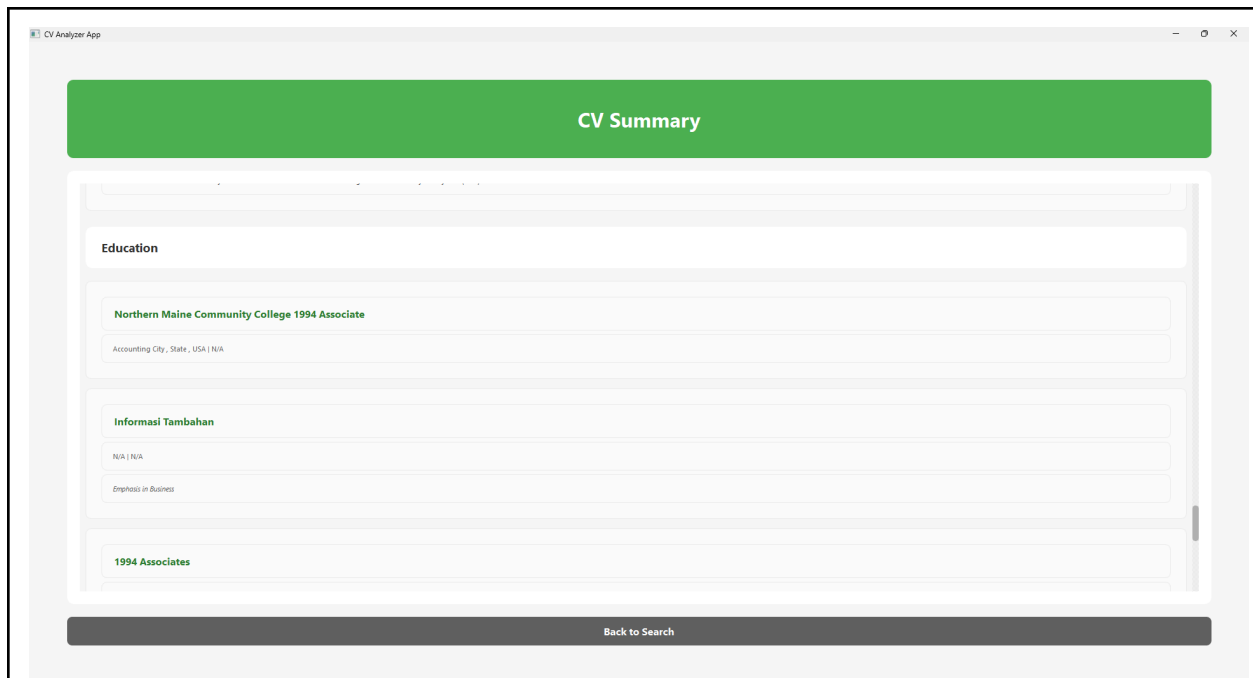
- Exact Match: scanned in 3.08s.
- Fuzzy Match: scanned in 4.08s.

The results are displayed in a white box with a green header bar containing the text "AR13L H3RFR150N". Below this, a green bar indicates "Matched keywords: 2". The results are listed as follows:

- accounting (similar: accounting) 3 occurrences
- accounting (similar: counting) 1 occurrence

At the bottom of the results box, there are two buttons: "Summary" (blue) and "View CV" (orange).



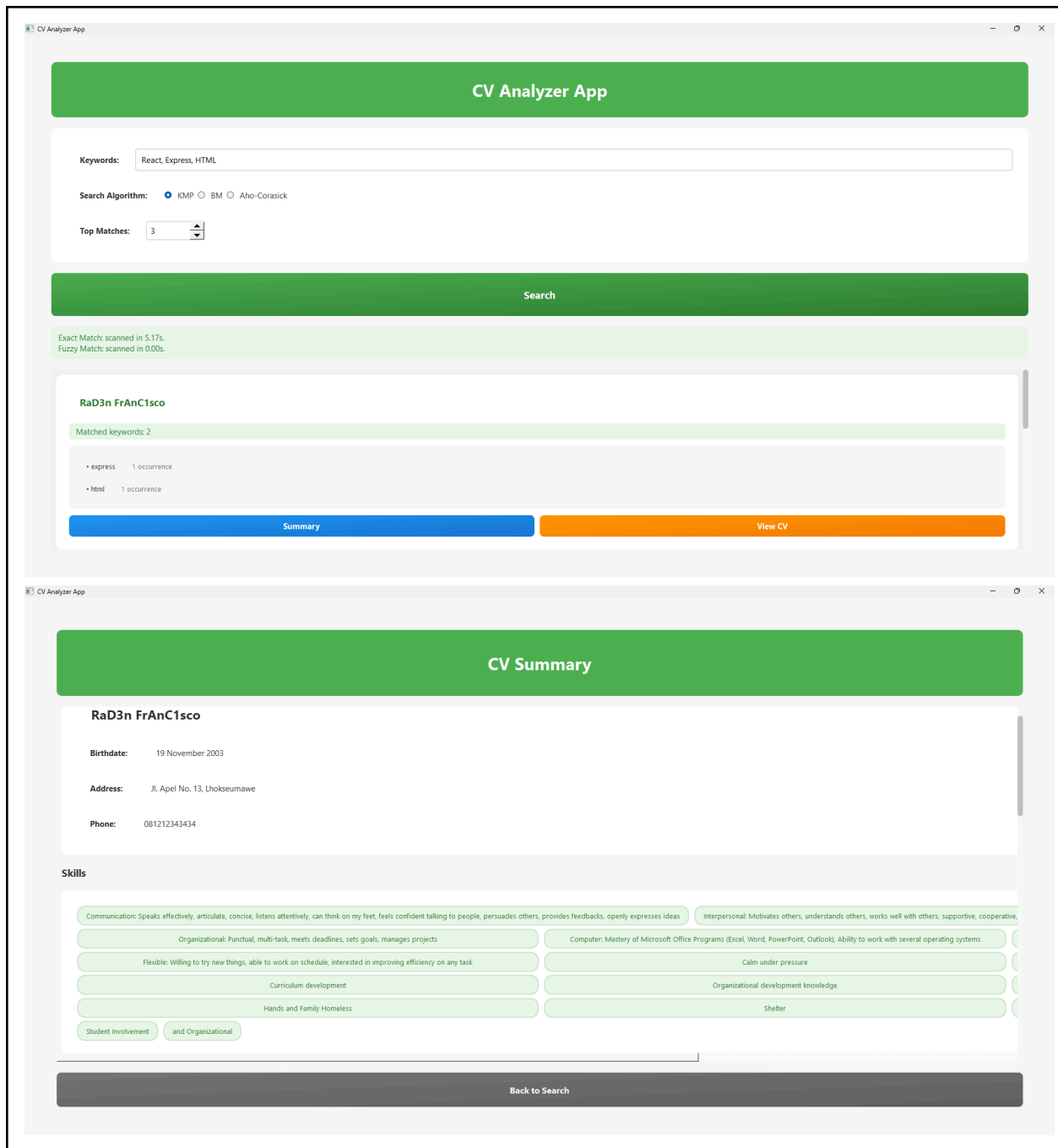


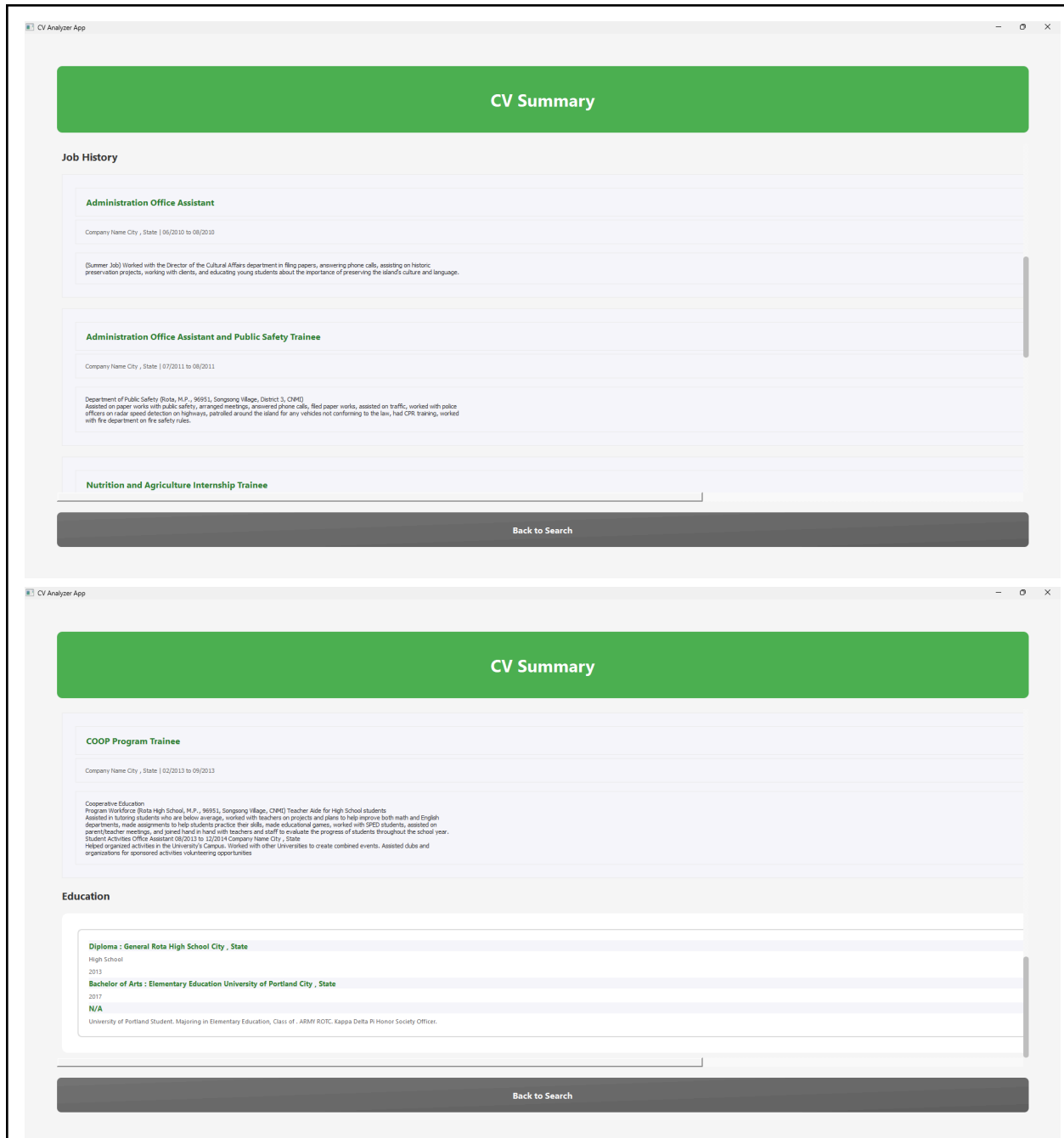
Analisis:

Analisis: Program berhasil melakukan pencarian terhadap keyword 'accounting' menggunakan algoritma KMP dengan menampilkan 2 hasil teratas. Pencarian berlangsung selama 3.08 detik dan mendapatkan 2 data yang relevan. Dapat terlihat bahwa menu CV summary bekerja dan dapat menampilkan detail summary dengan baik.

Test Case 4: (React, Express, HTML - KMP - 3)

Output:

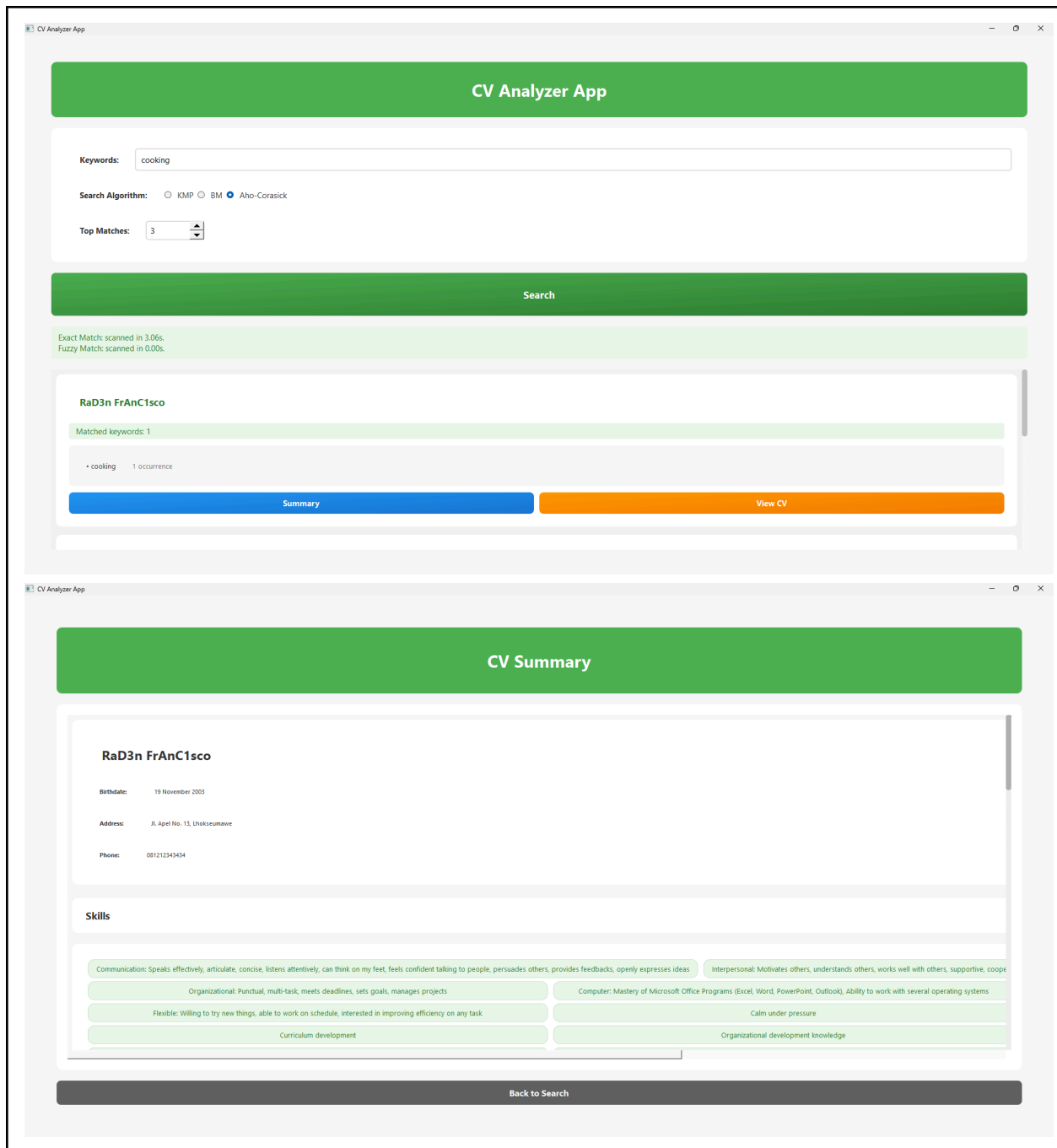




Analisis: Program berhasil melakukan pencarian terhadap 3 keywords menggunakan algoritma KMP dengan menampilkan 3 hasil teratas (pada dataset hanya ditemukan satu data yang cocok). Pencarian berlangsung selama 5.17 detik. Dapat terlihat bahwa menu CV summary bekerja dan dapat menampilkan detail summary dengan baik.

Test Case 5: (cooking - Aho Corasick - 3)

Output:



CV Analyzer App

CV Summary

Job History

Company Name City, State

Administration Office Assistant | 06/2010 to 08/2010

(Summer Job) Worked with the Director of the Cultural Affairs department in filing papers, answering phone calls, assisting on historic preservation projects, working with clients, and educating young students about the importance of preserving the island's culture and language. 2. Department of Public Safety (Rota, M.P., 96951, Songroing Village, District 3, CHMI)

Company Name City, State

Administration Office Assistant and Public Safety Trainee | 07/2011 to 08/2011

Assisted on paper works with public safety, arranged meetings, answered phone calls, filed paper works, assisted on traffic, worked with police officers on radar speed detection on highways, patrolled around the island for any vehicles not conforming to the law, had CPR training, worked with fire department on fire safety rules. 3. Northern Marianas College Internship Program (Rota, M.P., 96951, Highway, CHMI)

Back to Search

CV Analyzer App

CV Summary

Education

High School Diploma

General Rota High School City, State | 2013

Bachelor of Arts

Elementary Education University of Portland City, State | 2017

Informasi Tambahan

NIA | 2017

University of Portland Student, Majoring in Elementary Education, Class of 2017. ADHY ROTC, Kappa Delta Pi Honor Society Officer.

Back to Search

Analisis:

Analisis: Program berhasil melakukan pencarian terhadap keyword ‘cooking’ menggunakan algoritma aho corasick dengan menampilkan 3 hasil teratas. Pencarian berlangsung selama 3.06 detik dan mendapatkan 3 data yang relevan. Dapat terlihat bahwa menu CV summary bekerja dan dapat menampilkan detail summary dengan baik. Dari analisis waktu terhadap dua algoritma sebelumnya, didapati bahwa algoritma Aho Corasick memiliki waktu pencarian paling cepat dalam kasus ini.

BAB V

PENUTUP

5.1. Kesimpulan

Pada Tugas Besar 2 mata kuliah IF2211 Strategi Algoritma ini, kami berhasil mengembangkan aplikasi CV Analyzer untuk mempermudah seleksi kandidat berdasarkan kata kunci. Program ini memanfaatkan algoritma pencarian Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) untuk mencocokkan teks, serta algoritma Levenshtein Distance untuk fuzzy matching, sehingga memungkinkan pencarian yang efisien dan fleksibel. Selain itu, dikembangkan modul ekstraksi teks dari PDF menggunakan PyMuPDF dan parsing berbasis regex untuk menyusun data kandidat, termasuk pengalaman kerja yang ditampilkan secara terstruktur. Lalu, dikembangkan frontend berbasis PyQt5 yang dapat memvisualisasikan hasil pencarian dan ringkasan CV dengan antarmuka yang intuitif.

5.2. Saran

Saran untuk pengembangan selanjutnya, sebaiknya dapat dicoba juga untuk melakukan implementasi beberapa bonus seperti implementasi seperti Membuat bonus enkripsi data profil *applicant*.

5.3. Refleksi

Melalui proyek ini, kami memperoleh banyak pemahaman baru dan pemahaman mendalam mengenai implementasi algoritma Knuth Morris Pratt dan Boyer Moore. Proyek ini berhasil menunjukkan bagaimana strategi algoritma dapat diterapkan secara praktis dalam menyelesaikan permasalahan.

LAMPIRAN

Link Penting

Repository	https://github.com/RafaAbdussalam/Tubes3_RiceCooker
------------	---

Tabel Checkpoint

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

Rinaldi Munir. Pencocokan string (string matching) dengan algoritma brute force, KMP, Boyer-Moore. Diakses pada 14 Juni 2025 dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

Rinaldi Munir. Pencocokan string dengan regular expression (regex). Diakses pada 14 Juni 2025 dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf)

Rinaldi Munir. Program Dinamis (Bagian 1). Diakses pada 14 Juni 2025 dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-(2025)-Bagian1.pdf)

Rinaldi Munir. Program Dinamis (Bagian 2). Diakses pada 14 Juni 2025 dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/26-Program-Dinamis-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/26-Program-Dinamis-(2025)-Bagian2.pdf)

GeeksforGeeks. (n.d.). KMP Algorithm for Pattern Searching. Diakses 14 Juni 2025, dari <https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/>

GeeksforGeeks. (n.d.). Boyer Moore Algorithm for Pattern Searching. Diakses 14 Juni 2025, dari <https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/>

GeeksforGeeks. (n.d.). Regular Expression in Python. Diakses 14 Juni 2025, dari <https://www.geeksforgeeks.org/regular-expression-python-examples/>

GeeksforGeeks. (n.d.). Introduction to Levenshtein Distance. Diakses 14 Juni 2025, dari <https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/>