IF2211 – Strategi Algoritma
Laporan Tugas Besar 3 : Pattern Matching

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

Dipersiapkan oleh:

LinkedOut

Boye Mangaratua Ginting	(13523127)
Muhammad Aulia Azka	(13523137)
Muhammad Rizain Firdaus	(13523164)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132

2025

Daftar Isi

Daftar Gambar

1. Deskripsi Tugas

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, kami diminta untuk mengimplementasikan sistem yang dapat melakukan deteksi informasi pelamar berbasis dokumen CV digital. Metode yang akan digunakan untuk melakukan deteksi pola dalam CV adalah Boyer-Moore 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.

2. Landasan Teori

2.1. Algoritma Knuth-Morris-Prath

Algoritma Knuth-Morris-Pratt, atau yang lebih dikenal sebagai algoritma KMP, adalah algoritma pencocokan string (string matching) yang efisien untuk mencari kemunculan sebuah pola (pattern) dalam teks (text). Algoritma ini dikembangkan oleh Donald Knuth, Vaughan Pratt, dan James H. Morris pada tahun 1977. KMP memanfaatkan informasi dari kegagalan pencocokan sebelumnya untuk menghindari pemeriksaan ulang karakter yang sudah diketahui tidak cocok, sehingga meningkatkan efisiensi dibandingkan pendekatan naif (brute force).

Prinsip Kerja Algoritma Knuth-Morris-Prath:

- Algoritma KMP menggunakan tabel prefix (juga disebut tabel failure function atau partial match table) untuk menyimpan informasi tentang panjang prefix terpanjang yang juga merupakan suffix untuk setiap posisi dalam pola.
- Tabel ini memungkinkan algoritma untuk melompati beberapa karakter dalam teks ketika terjadi ketidakcocokan, tanpa perlu memeriksa ulang karakter yang sudah cocok.
- Kompleksitas waktu algoritma KMP adalah O(n + m), di mana n adalah panjang teks dan m adalah panjang pola. Kompleksitas ruangnya adalah O(m) untuk menyimpan tabel prefix.

2.2. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah algoritma pencocokan string lain yang sangat efisien, dikembangkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Berbeda dengan algoritma KMP yang memeriksa teks dari kiri ke kanan, Boyer-Moore memeriksa teks dari kanan ke kiri (*right-to-left*). Algoritma ini dikenal karena kemampuannya untuk melompati sejumlah besar karakter dalam teks, terutama ketika pola tidak sering muncul dalam teks.

Prinsip Kerja Algoritma Boyer-Moore:

- Algoritma ini menggunakan dua heuristik utama:
    1. Bad Character Heuristic: Ketika terjadi ketidakcocokan antara karakter teks dan pola, algoritma menggunakan tabel *bad character* untuk menentukan seberapa jauh pola dapat digeser berdasarkan karakter yang tidak cocok.
    2. Good Suffix Heuristic: Jika sebagian pola sudah cocok tetapi gagal di karakter tertentu, algoritma menggunakan tabel *good suffix* untuk menentukan geseran berdasarkan bagian pola yang sudah cocok.
- Dengan kedua heuristik ini, Boyer-Moore dapat melompati banyak karakter, terutama jika alfabet yang digunakan besar.

- Kompleksitas waktu rata-rata adalah O(n/m) dalam kasus terbaik (ketika pola jarang muncul), tetapi dalam kasus terburuk tetap O(n × m). Kompleksitas ruang adalah O(σ + m), di mana σ adalah ukuran alfabet.

3. Analisis Pemecahan Masalah

3.1. Langkah-Langkah Pemecahan Masalah

3.1.1. Analisis Kebutuhan Sistem

Langkah awal adalah menganalisis secara mendalam spesifikasi yang diberikan. Proses ini bertujuan untuk mengidentifikasi semua fitur utama yang harus dimiliki oleh aplikasi, antara lain:

1) Ekstraksi Data CV: Kemampuan untuk mengurai (*parsing*) dokumen CV berformat PDF dan mengekstrak informasi penting seperti data diri, ringkasan, pengalaman kerja, pendidikan, dan keahlian menggunakan *Regular Expression* (Regex).

2) Manajemen Database: Kebutuhan untuk merancang dan mengimplementasikan basis data MySQL untuk menyimpan profil pelamar dan detail lamaran secara terstruktur.

3) Antarmuka Pengguna (GUI): Perancangan antarmuka desktop yang intuitif menggunakan Python, memungkinkan pengguna mengunggah CV, melakukan pencarian, memilih algoritma, dan melihat hasil.

4) Implementasi Algoritma Pencarian: Kewajiban untuk mengimplementasikan algoritma *exact matching* (Knuth-Morris-Pratt dan Boyer-Moore) dan algoritma *fuzzy matching* (Levenshtein Distance).

5) Fitur Tampilan Hasil: Kemampuan untuk menampilkan hasil pencarian yang relevan, diurutkan berdasarkan jumlah kecocokan, menampilkan ringkasan profil, dan menyajikan waktu eksekusi pencarian.

3.1.2. Perancangan Sistem

Perancangan Database: Merancang skema tabel untuk database MySQL. Dua tabel utama diidentifikasi: `ApplicantProfile` untuk menyimpan data hasil ekstraksi (nama, email, ringkasan, keahlian, dll.) dan `ApplicationDetail` untuk menyimpan data terkait aplikasi (posisi yang dilamar, path file CV).

Perancangan Arsitektur Aplikasi: Membagi aplikasi menjadi beberapa modul utama yang saling berinteraksi:

- Modul GUI (`gui`): Bertanggung jawab atas semua interaksi dengan pengguna.
- Modul Ekstraksi (`extract`): Berisi logika untuk membaca file PDF dan mengekstrak informasi menggunakan Regex.
- Modul Pencocokan Pola (`pattern`): Implementasi algoritma KMP, Boyer-Moore, dan Levenshtein.

- Modul Database (`database_connector`): Bertugas sebagai jembatan antara aplikasi dan database MySQL.

Perancangan Alur Kerja Pengguna: Memetakan bagaimana pengguna akan berinteraksi dengan sistem, mulai dari mengunggah CV, data diproses dan disimpan, hingga melakukan pencarian dan melihat hasilnya.

3.1.3. Implementasi

Implementasi dilakukan secara modular sesuai dengan arsitektur yang telah dirancang.

- Pengembangan script untuk inisialisasi database (`init_db.py`).
- Pembuatan fungsi-fungsi untuk interaksi database (`query_service.py`).
- Implementasi kelas dan fungsi untuk ekstraksi CV dan Regex (`cv_extractor.py`, `info_extractor.py`).
- Pengkodean algoritma KMP, BM, dan Levenshtein (`matching.py`).
- Pembangunan antarmuka pengguna grafis (`main_window.py`) dan mengintegrasikan semua modul menjadi satu aplikasi yang utuh.

3.1.4. Pengujian dan Evaluasi

Setelah implementasi selesai, dilakukan serangkaian pengujian untuk memastikan sistem berjalan sesuai spesifikasi.

- Pengujian Ekstraksi: Menguji kemampuan sistem dalam mengekstrak informasi dari berbagai format CV PDF secara akurat.
- Pengujian Algoritma: Memverifikasi bahwa algoritma KMP, BM, dan Levenshtein memberikan hasil pencarian yang benar dan akurat.
- Pengujian Fungsionalitas: Memastikan semua fitur seperti unggah, simpan ke database, pencarian, dan penampilan hasil berfungsi tanpa kesalahan.
- Evaluasi Kinerja: Mengukur waktu yang dibutuhkan untuk proses pencarian (baik exact maupun fuzzy) sebagai salah satu output yang ditampilkan kepada pengguna.

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

Untuk mengaplikasikan algoritma pencocokan string seperti Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM), masalah fungsional "pencarian kata kunci dalam CV" harus dipetakan menjadi elemen-elemen komputasi yang dapat diolah oleh algoritma tersebut. Secara fundamental, algoritma pencocokan string beroperasi pada dua input utama: sebuah **`text`** (teks sumber yang panjang) dan sebuah **`pattern`** (pola atau string pendek yang dicari).

Proses pemetaan masalah ini melibatkan identifikasi dan persiapan kedua elemen tersebut dari data yang ada dalam sistem.

3.2.1. Identifikasi dan Standardisasi Pattern

Elemen `pattern` dalam konteks sistem ini secara langsung dipetakan dari input pengguna.

- **Sumber:** `Pattern` adalah kata kunci (keyword) yang dimasukkan oleh pengguna melalui kolom pencarian pada antarmuka grafis (GUI).
- **Proses:** Setiap kata kunci yang dimasukkan—baik tunggal maupun majemuk (dipisahkan oleh koma)—diperlakukan sebagai sebuah `pattern` individual.
- **Standardisasi:** Sebelum digunakan dalam proses pencarian, setiap `pattern` akan melalui tahap pra-pemrosesan sederhana, yaitu konversi seluruh karakter menjadi huruf kecil (*lowercase*). Langkah ini bertujuan untuk memastikan proses pencocokan bersifat *case-insensitive*, sehingga pencarian "Python" akan memberikan hasil yang sama dengan "python".

3.2.2. Agregasi dan Normalisasi Text

Elemen `text` merupakan representasi dari konten CV pelamar yang akan menjadi target pencarian. Alih-alih menggunakan teks mentah hasil ekstraksi PDF secara langsung, sistem ini menggunakan pendekatan yang lebih terstruktur untuk membentuk `text` yang relevan dan bersih.

- **Sumber:** `Text` tidak berasal dari satu sumber tunggal, melainkan merupakan hasil agregasi (penggabungan) dari beberapa kolom data yang tersimpan dalam tabel `ApplicantProfile` di database. Kolom-kolom yang digabungkan adalah yang paling relevan untuk pencarian kualifikasi, yaitu:
  - `summary`: Ringkasan profil pelamar.
  - `skills`: Daftar keahlian yang dimiliki.
  - `experience`: Deskripsi pengalaman kerja.
  - `education`: Riwayat pendidikan.
- **Proses:** Untuk setiap pelamar dalam database, konten dari kolom-kolom di atas digabungkan menjadi sebuah string tunggal yang panjang. Proses ini menciptakan sebuah "dokumen virtual" untuk setiap pelamar yang berisi informasi paling krusial.
- **Normalisasi:** Serupa dengan `pattern`, string `text` yang telah diagregasi juga melalui proses normalisasi. Proses ini mencakup konversi ke huruf kecil (*lowercase*) dan pembersihan dari karakter-karakter non-alfanumerik serta spasi berlebih. Tujuannya adalah untuk menciptakan sebuah teks sumber yang konsisten dan bersih, sehingga memaksimalkan akurasi algoritma pencocokan.

3.2.3. *Mekanisme Eksekusi Algoritma*

Setelah `pattern` dan `text` didefinisikan, proses pencocokan dapat dieksekusi sebagai berikut:

- Aplikasi mengambil seluruh data pelamar dari database.
- Pengguna memasukkan satu atau lebih `pattern` (kata kunci) dan memilih algoritma (KMP atau BM).
- Untuk setiap pelamar, aplikasi menyusun `text` sumber dengan menggabungkan dan menormalisasi data sesuai penjelasan di atas.
- Aplikasi kemudian mengiterasi setiap `pattern` yang dimasukkan pengguna.
- Algoritma yang dipilih (`kmp_search` atau `boyer_moore_search`) dieksekusi dengan `text` milik pelamar sebagai argumen pertama dan `pattern` (kata kunci) sebagai argumen kedua.
- Fungsi algoritma akan mengembalikan jumlah kemunculan `pattern` di dalam `text`.
- Jumlah total kemunculan dari semua kata kunci akan diakumulasikan untuk setiap pelamar. Nilai total inilah yang kemudian digunakan sebagai metrik relevansi untuk mengurutkan hasil pencarian, di mana pelamar dengan jumlah kecocokan tertinggi akan ditampilkan di urutan paling atas.

3.3. *Fitur Fungsional dan Arsitektur GUI yang Dibangun*

3.3.1. *Fitur Fungsional*

Sistem yang dibangun memiliki serangkaian fitur utama yang dirancang untuk memenuhi kebutuhan proses penyaringan pelamar kerja secara efisien. Fitur-fitur tersebut adalah:

- Pengunggahan dan Pemrosesan CV: Sistem menyediakan fungsionalitas bagi pengguna untuk mengunggah dokumen CV dalam format PDF. Setelah diunggah, sistem secara otomatis menjalankan proses ekstraksi teks dan informasi terstruktur (seperti keahlian, pengalaman, dan pendidikan) untuk kemudian disimpan ke dalam basis data.
- Manajemen Basis Data Pelamar: Setiap CV yang berhasil diproses akan disimpan sebagai profil pelamar yang utuh dalam database MySQL. Sistem ini berfungsi sebagai repositori terpusat yang dapat diakses untuk pencarian.

- Pencarian Komprehensif: Fitur inti dari sistem ini adalah pencarian pelamar berdasarkan kata kunci.
    - Pencarian *Exact*: Pengguna dapat memilih antara algoritma Knuth-Morris-Pratt (KMP) atau Boyer-Moore (BM) untuk melakukan pencarian kata kunci yang persis.
    - Pencarian *Fuzzy*: Apabila kata kunci tidak ditemukan melalui pencarian *exact*, sistem secara otomatis akan melakukan pencarian kembali menggunakan algoritma Levenshtein Distance untuk menemukan kecocokan yang mendekati, berguna untuk mengatasi kesalahan ketik.
- Tampilan Hasil yang Relevan: Hasil pencarian ditampilkan dalam bentuk daftar yang diurutkan berdasarkan relevansi (jumlah kata kunci yang cocok paling banyak). Pengguna juga dapat menentukan jumlah maksimal hasil yang ingin ditampilkan untuk membatasi output.

- Ringkasan Profil Pelamar: Dari daftar hasil pencarian, pengguna dapat memilih salah satu pelamar untuk melihat halaman ringkasan (summary) yang lebih detail. Halaman ini menampilkan semua informasi penting yang telah diekstrak dari CV dan menyediakan opsi untuk membuka file PDF asli dari pelamar tersebut.

- Analisis Waktu Eksekusi: Untuk memberikan wawasan performa, sistem menampilkan total waktu yang dibutuhkan untuk setiap proses pencarian, baik untuk pencarian *exact* maupun *fuzzy* (jika dijalankan).

3.3.2. Arsitektur dan Desain Antarmuka (GUI)

Antarmuka sistem ini dibangun sebagai aplikasi desktop menggunakan bahasa pemrograman Python dengan pustaka CustomTkinter untuk menciptakan tampilan yang modern dan responsif. Arsitektur GUI dirancang secara modular untuk memisahkan fungsionalitas dan meningkatkan kemudahan penggunaan.

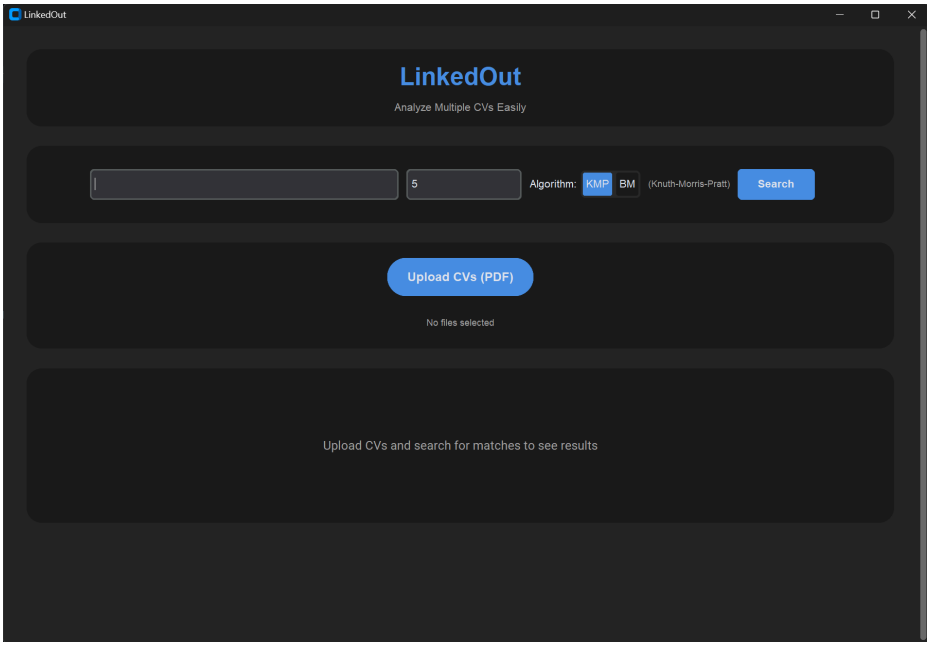
Struktur utama GUI terdiri dari komponen-komponen berikut:

- Jendela Utama (Main Window): Merupakan tampilan utama aplikasi yang terbagi menjadi tiga area utama:
    - Frame Kontrol (Atas): Area ini berisi semua alat interaksi untuk pengguna, seperti tombol "Upload CV", kolom input untuk kata kunci pencarian, menu *dropdown* untuk memilih algoritma (KMP/BM), dan kolom untuk menentukan jumlah hasil.
    - Frame Hasil (Tengah): Sebuah area yang dapat di-scroll (*scrollable frame*) yang secara dinamis menampilkan daftar pelamar hasil pencarian. Setiap entri dalam daftar ini menampilkan informasi ringkas seperti nama pelamar dan jumlah kecocokan, beserta tombol "Lihat Detail".
- Jendela Ringkasan (Summary Window): Sebuah jendela baru (*toplevel window*) yang akan muncul ketika pengguna mengklik tombol "Lihat Detail". Jendela ini didedikasikan untuk menampilkan seluruh

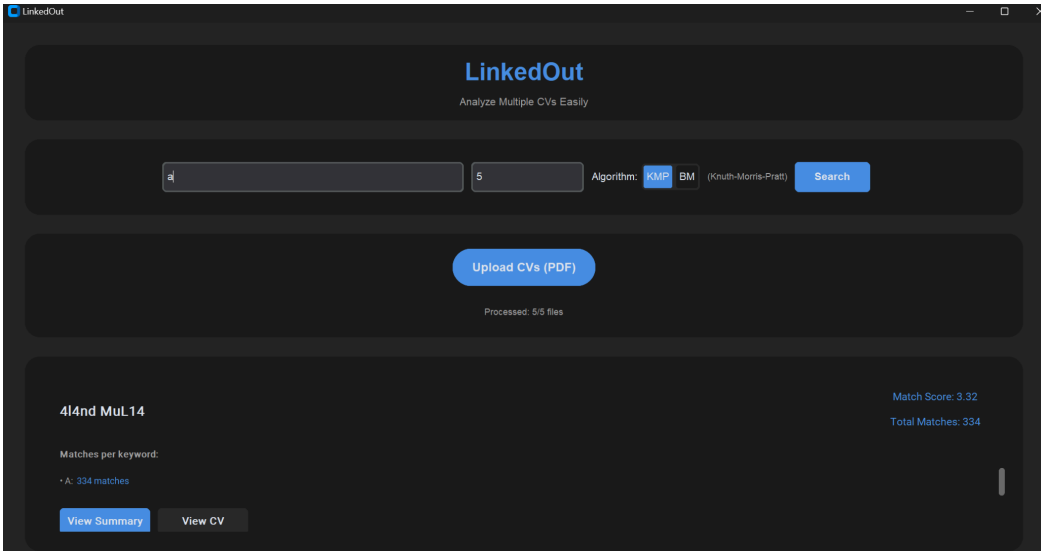
informasi terstruktur dari satu pelamar yang dipilih, serta tombol untuk membuka file CV asli.

3.4. Contoh Ilustrasi Kasus

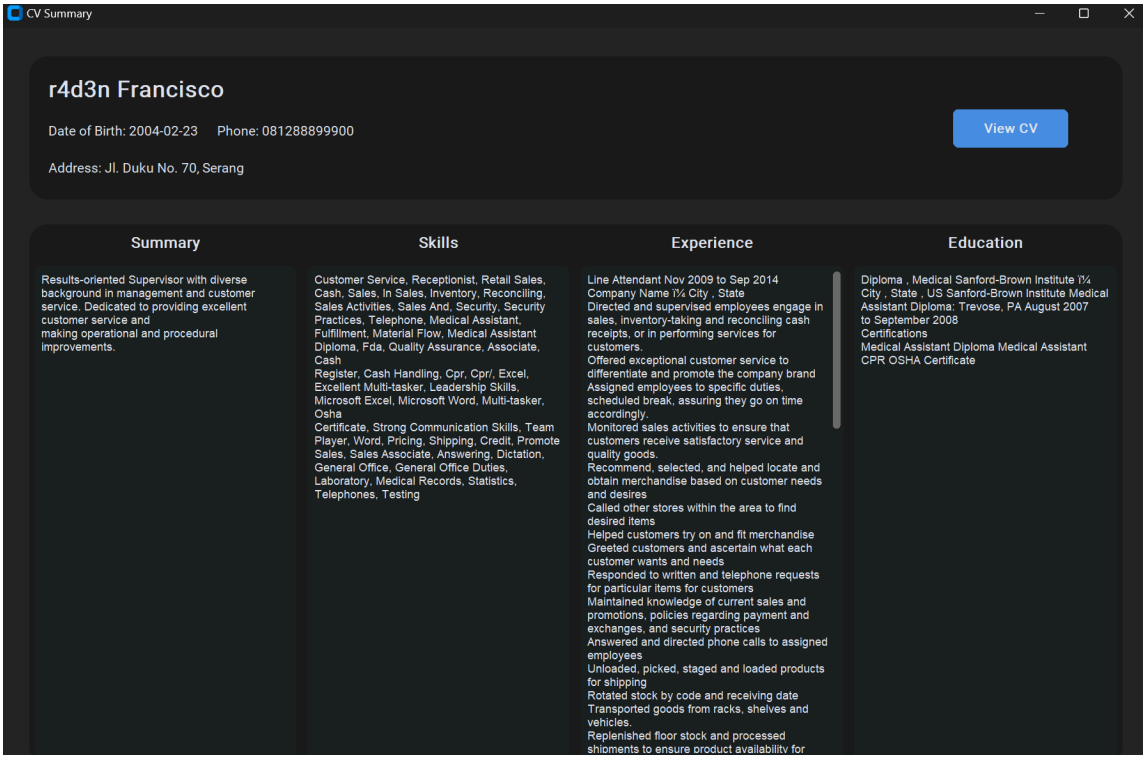
Berikut adalah beberapa gambar mengenai program kami :



Gambar 3.3.1. Tampilan Mula-Mula



Gambar 3.3.2. Tampilan Output Program



Gambar 3.3.2. Tampilan Summary

4. Implementasi dan Pengujian

4.1. Implementasi

4.1.1. regex.py

```
import re

def locate_section(content, search_terms):
    """
    Locate the beginning of a section using provided search terms.
    Returns matching term and its position.
    """
    for term in search_terms:
        try:
            regex_pattern = r'^\s*' + re.escape(term) + r'\s*$'
            found_match = re.search(regex_pattern, content,
re.IGNORECASE | re.MULTILINE)
            if found_match:
                return term, found_match.start()
        except re.error:
            continue
    return None, -1

def retrieve_section_content(content, section_keywords,
boundary_headings):
    """
    Retrieves content for a specific section, ending at the next
recognized heading.
    """
    _, section_start = locate_section(content, section_keywords)
    if section_start == -1:
        return "Not Found"

    section_end = len(content)

    # Locate the beginning of the subsequent section to define current
section boundary
    # Search within text following the current section start
    remaining_content = content[section_start + 1:]

    for header in boundary_headings:
        # Skip keywords belonging to the current section
        if header.lower() in [keyword.lower() for keyword in
section_keywords]:
            continue

        _, next_header_pos = locate_section(remaining_content,
[header])
        if next_header_pos != -1:
            # Convert relative position to absolute position
            absolute_position = section_start + 1 + next_header_pos
            section_end = min(section_end, absolute_position)

    # Get content from end of header line to beginning of next section
    header_line_match = re.search(r'.*', content[section_start:])
    if header_line_match:
        content_start = section_start + header_line_match.end()
        return content[content_start:section_end].strip()

    return "Not Found"

def parse_cv_sections(content):
    """
    Parses all essential sections (Summary, Skills, Experience,
Education) from CV content.
    """
    section_mappings = {
        'summary': ['professional summary', 'executive profile',
'career overview', 'executive summary','summary', 'overview',
'profile', 'objective'],
```

```
        'skills': [ 'areas of expertise', 'skill highlights', 'core
strengths', 'core qualifications','skills', 'abilities',
'technologies'],
        'experience': ['professional experience', 'work experience',
'teaching experience','experience', 'work history', 'employment
history'],
        'education': [ 'education and training','education',
'qualifications']
    }

    recognized_headers = [ # Complete Header Map
        # Summary variants
        'Executive Profile','career overview', 'executive summary',
'Summary', 'Overview', 'Profile', 'Objective',
        # Skills variants
        'Areas of Expertise','Skill Highlights', 'core strengths',
'core qualifications','Skills', 'Abilities', 'Technologies',
        # Experience variants
         'Work experience', 'Teaching experience','Professional
Experience', 'Experience','Work History', 'Employment History',
        # Education variants
        'Education and Training','Education',
        # Termination markers
        'Accomplishments', 'Highlights', 'Additional Information',
'References', 'Website and Links', 'Affiliations'
    ]

    parsed_sections = {}
    for section_key, keyword_list in section_mappings.items():
        parsed_sections[section_key] =
retrieve_section_content(content, keyword_list, recognized_headers)

    return parsed_sections
```

4.1.2. cv_extractor.py

```
import fitz
import re

class CVExtractor:
    def __init__(self, file_path):
        self.file_path = file_path
        self._raw_content = ""
        self._processed_content = ""

    def retrieve_raw_text(self):
        """Return the raw text content from the PDF"""
        if not self._raw_content:
            self._extract_text_from_pdf()
        return self._raw_content

    def retrieve_cleaned_text(self):
        """Return the processed text as a continuous string"""
        if not self._processed_content:
            self._convert_to_continuous_text()
        return self._processed_content

    def get_file_path(self):
        """Return the current PDF file path"""
        return self.file_path

    def update_file_path(self, path):
        """Update the PDF file path and reset cached content"""
        self.file_path = path
        self._raw_content = ""
        self._processed_content = ""

    def _extract_text_from_pdf(self):
        """Extract text content from all PDF pages"""
```

```
        document = fitz.open(self.file_path)
        text_content = ""
        for page_num in range(len(document)):
            page = document[page_num]
            text_content += page.get_text()
        document.close()
        self._raw_content = text_content

    def _convert_to_continuous_text(self):
        """Process text to lowercase continuous string without
punctuation"""
        if not self._raw_content:
            self._extract_text_from_pdf()

        # Strip punctuation marks using regex
        no_punct_text = re.sub(r'[^\w\s]', '', self._raw_content)
        # Normalize whitespace and convert to lowercase
        normalized_text = re.sub(r'\s+', ' ',
no_punct_text).lower().strip()
        self._processed_content = normalized_text

    def process(self):
        """Execute the complete text extraction and processing
workflow"""
        self._extract_text_from_pdf()
        self._convert_to_continuous_text()
```

Kelas CVExtractor dirancang untuk mengekstrak dan memproses teks dari file PDF, seperti CV, dengan alur yang efisien dan terstruktur. Proses dimulai dengan inisialisasi objek, di mana path file PDF disimpan sebagai atribut, dan dua variabel privat, _raw_content dan _processed_content, diatur sebagai string kosong untuk menyimpan teks mentah dan teks yang telah diproses. Metode retrieve_raw_text mengembalikan teks mentah dari PDF dengan memeriksa apakah _raw_content sudah terisi; jika belum, metode privat _extract_text_from_pdf dipanggil untuk mengekstrak teks dari semua halaman PDF menggunakan pustaka fitz, lalu menyimpannya ke _raw_content. Untuk teks yang telah diproses, metode retrieve_cleaned_text mengembalikan teks yang telah dinormalisasi dengan huruf kecil, tanpa tanda baca, dan spasi yang distandarisasi; jika _processed_content kosong, metode _convert_to_continuous_text dipanggil untuk memproses teks mentah dengan menghapus tanda baca menggunakan regex, menormalisasi spasi, dan mengubahnya ke huruf kecil. Metode get_file_path hanya mengembalikan path file PDF saat ini, sedangkan update_file_path memperbarui path file dan mengosongkan kembali _raw_content dan _processed_content untuk memastikan data lama tidak digunakan. Terakhir, metode process menjalankan alur lengkap dengan memanggil _extract_text_from_pdf dan _convert_to_continuous_text secara berurutan. Pendekatan lazy loading digunakan pada metode pengambilan teks untuk menghindari pemrosesan berulang, sehingga meningkatkan efisiensi.

4.1.3. bm.py

```
def boyer_moore_search(text, pattern):
    """
    Melakukan pencarian string menggunakan algoritma Boyer-Moore
```

```
    dengan Bad Character Heuristic dan Good Suffix Heuristic.
       Mengembalikan list berisi indeks awal dari semua kemunculan
pattern.
    """
    m = len(pattern)
    n = len(text)
    if m == 0:
        return []
    if n < m:
        return []

    matches = []

    # --- Preprocessing untuk Bad Character Heuristic ---
    bad_char = {}
    for i in range(m):
        bad_char[pattern[i]] = i

    # --- Preprocessing untuk Good Suffix Heuristic ---
    s = [0] * (m + 1)
    f = [0] * (m + 1)

    i = m
    j = m + 1
    f[i] = j
    while i > 0:
        while j <= m and pattern[i - 1] != pattern[j - 1]:
            if s[j] == 0:
                s[j] = j - i
            j = f[j]
        i -= 1
        j -= 1
        f[i] = j

    j = f[0]
    for i in range(m + 1):
        if s[i] == 0:
            s[i] = j
        if i == j:
            j = f[j]

    # --- Proses Pencarian ---
    shift = 0
    while shift <= n - m:
        j = m - 1
        while j >= 0 and pattern[j] == text[shift + j]:
            j -= 1

        if j < 0:
            matches.append(shift)
                # Geser berdasarkan Good Suffix Rule untuk menemukan
kemunculan berikutnya
            shift += s[0]
        else:
             # Geser berdasarkan nilai maksimum dari Bad Character dan
Good Suffix
            char_text = text[shift + j]
            bad_char_shift = j - bad_char.get(char_text, -1)
            good_suffix_shift = s[j + 1]

            shift += max(bad_char_shift, good_suffix_shift)

    return matches
```

Fungsi boyer_moore_search mengimplementasikan algoritma pencarian string Boyer-Moore untuk menemukan semua kemunculan pola (pattern) dalam teks (text) menggunakan dua heuristik: Bad Character dan Good Suffix, lalu mengembalikan daftar indeks awal kemunculan pola. Proses dimulai dengan memeriksa panjang pola (m) dan teks (n); jika pola kosong atau teks lebih pendek dari pola, fungsi mengembalikan daftar

kosong. Selanjutnya, tahap preprocessing untuk Bad Character Heuristic dilakukan dengan membuat kamus bad_char yang menyimpan indeks terakhir setiap karakter dalam pola untuk menentukan pergeseran saat ketidakcocokan terjadi. Untuk Good Suffix Heuristic, dua array s dan f dibuat untuk menghitung pergeseran berdasarkan sufiks yang cocok; array f digunakan untuk melacak batas sufiks, dan array s menyimpan jumlah pergeseran minimum berdasarkan sufiks yang cocok atau batas sufiks, dengan logika iterasi mundur untuk mengisi nilai-nilai ini. Dalam proses pencarian, variabel shift menunjukkan posisi awal pola dalam teks, dan perbandingan dilakukan dari kanan ke kiri (indeks j dari m-1 ke 0) untuk mencocokkan karakter pola dengan teks. Jika semua karakter cocok (j < 0), indeks shift ditambahkan ke daftar matches, dan pergeseran dilakukan berdasarkan nilai s[0] dari Good Suffix Rule untuk mencari kemunculan berikutnya. Jika terjadi ketidakcocokan, pergeseran dihitung sebagai maksimum antara Bad Character Shift (berdasarkan karakter teks yang tidak cocok dan kamus bad_char) dan Good Suffix Shift (dari array s[j+1]), memastikan pergeseran optimal untuk melewati bagian teks yang tidak relevan. Proses ini berulang hingga shift melebihi panjang teks dikurangi panjang pola, dan daftar matches dikembalikan sebagai hasil akhir. Algoritma ini efisien karena meminimalkan perbandingan karakter dengan memanfaatkan informasi ketidakcocokan untuk pergeseran besar.

4.1.4. kmp.py

```
def kmp_search(text, pattern):
    """
    Melakukan pencarian string menggunakan algoritma
Knuth-Morris-Pratt.
    Mengembalikan list berisi indeks awal dari semua kemunculan
pattern.
    """
    def compute_lps(pattern):
        # Longest Proper Prefix which is also Suffix
        lps = [0] * len(pattern)
        length = 0
        i = 1
        while i < len(pattern):
            if pattern[i] == pattern[length]:
                length += 1
                lps[i] = length
                i += 1
            else:
                if length != 0:
                    length = lps[length - 1]
                else:
                    lps[i] = 0
                    i += 1
        return lps

    if not pattern or not text:
        return []

    lps = compute_lps(pattern)
    matches = []
    i = 0  # pointer untuk text
    j = 0  # pointer untuk pattern
    while i < len(text):
        if pattern[j] == text[i]:
            i += 1
            j += 1
```

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

Fungsi kmp_search mengimplementasikan algoritma Knuth-Morris-Pratt (KMP) untuk mencari semua kemunculan pola (`pattern`) dalam teks (`text`) dan mengembalikan daftar indeks awal kemunculan tersebut. Algoritma ini bekerja secara efisien dengan menghindari perbandingan ulang karakter yang sudah cocok melalui penggunaan tabel Longest Proper Prefix which is also Suffix (LPS). Alur dimulai dengan memeriksa apakah teks atau pola kosong; jika ya, fungsi mengembalikan daftar kosong. Fungsi pembantu `compute_lps` dipanggil untuk membangun tabel LPS, yang menghitung panjang prefiks terpanjang yang juga merupakan sufiks untuk setiap posisi dalam pola. Proses ini dilakukan dengan mengiterasi pola mulai dari indeks 1, membandingkan karakter saat ini dengan karakter pada posisi `length` (panjang prefiks/sufiks saat ini); jika cocok, `length` bertambah dan disimpan di `lps`, jika tidak, `length` dikurangi berdasarkan nilai `lps` sebelumnya hingga mencapai 0 atau menemukan kecocokan. Dalam proses pencarian utama, dua pointer digunakan: `i` untuk teks dan `j` untuk pola. Jika karakter pada `text[i]` dan `pattern[j]` cocok, kedua pointer maju. Jika `j` mencapai panjang pola, kecocokan ditemukan, indeks awal (`i - j`) ditambahkan ke daftar `matches`, dan `j` diatur ulang ke `lps[j - 1]` untuk mencari kemunculan berikutnya. Jika karakter tidak cocok, `j` dikurangi ke `lps[j - 1]` jika `j` bukan 0, atau `i` maju jika `j` adalah 0, memungkinkan algoritma melewati karakter teks yang tidak relevan. Proses berulang hingga `i` melewati panjang teks, dan daftar `matches` dikembalikan. Algoritma ini efisien karena tabel LPS meminimalkan perbandingan berulang, menjadikannya cocok untuk pencarian teks seperti dalam dokumen atau CV.

4.1.5. fuzzy.py

```
def levenshtein_distance(s1: str, s2: str) -> int:
    """
    Menghitung jarak Levenshtein antara dua string.
    Jarak Levenshtein adalah jumlah minimum operasi single-character
edit
    (insertions, deletions, substitutions) yang diperlukan untuk
mengubah satu string menjadi string lainnya.
    """
    if len(s1) < len(s2):
        return levenshtein_distance(s2, s1)

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

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
```

```
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions,
substitutions))
        previous_row = current_row

    return previous_row[-1]

def calculate_similarity(s1: str, s2: str) -> float:
    """
    Menghitung tingkat kemiripan antara dua string menggunakan
Levenshtein Distance.
    Mengembalikan nilai antara 0 (tidak mirip) sampai 1 (identik).
    """
    if not s1 or not s2:
        return 0.0

    max_len = max(len(s1), len(s2))
    if max_len == 0:
        return 1.0

    distance = levenshtein_distance(s1, s2)
    return 1.0 - (distance / max_len)

def fuzzy_search(text: str, pattern: str, threshold: float = 0.8) ->
list[tuple[int, float]]:
    """
    Melakukan pencarian fuzzy menggunakan Levenshtein Distance.

    Args:
        text: Teks yang akan dicari
        pattern: Pola yang dicari
        threshold: Ambang batas kemiripan (0.0 - 1.0)

    Returns:
        List of tuples (index, similarity_score) untuk setiap
kemunculan yang memenuhi threshold
    """
    text = text.lower()
    pattern = pattern.lower()
    matches = []

    words = text.split()

    for i, word in enumerate(words):
        similarity = calculate_similarity(word, pattern)
        if similarity >= threshold:
            start_pos = text.find(word)
            matches.append((start_pos, similarity))

    return sorted(matches, key=lambda x: x[1], reverse=True)

def fuzzy_search_all(text: str, patterns: list[str], threshold: float
= 0.8) -> dict[str, list[tuple[int, float]]]:
    """
    Melakukan pencarian fuzzy untuk multiple patterns.

    Args:
        text: Teks yang akan dicari
        patterns: List of patterns yang dicari
        threshold: Ambang batas kemiripan (0.0 - 1.0)

    Returns:
        Dictionary dengan key pattern dan value list of tuples (index,
similarity_score)
    """
    results = {}
    for pattern in patterns:
        matches = fuzzy_search(text, pattern, threshold)
        if matches:
            results[pattern] = matches
    return results
```

Fungsi-fungsi yang diberikan (levenshtein_distance, calculate_similarity, fuzzy_search, dan fuzzy_search_all) dirancang untuk melakukan pencarian fuzzy pada teks menggunakan jarak Levenshtein, yang mengukur kemiripan antar string. Alur algoritmanya dimulai dengan levenshtein_distance, yang menghitung jumlah minimum operasi edit (penyisipan, penghapusan, substitusi) untuk mengubah string s1 menjadi s2. Fungsi ini menggunakan pendekatan pemrograman dinamis dengan menginisialisasi baris sebelumnya (previous_row) sebagai rentang angka dari 0 hingga panjang s2 ditambah satu, lalu mengiterasi setiap karakter di s1 untuk membangun baris saat ini (current_row). Untuk setiap pasangan karakter, fungsi menghitung biaya operasi penyisipan (dari previous_row[j+1] + 1), penghapusan (dari current_row[j] + 1), dan substitusi (dari previous_row[j] ditambah 1 jika karakter berbeda), lalu mengambil nilai minimumnya. Hasil akhir adalah nilai terakhir di baris terakhir, yaitu jarak Levenshtein. Fungsi calculate_similarity memanfaatkan jarak ini untuk menghitung skor kemiripan antara dua string dengan rumus 1.0 - (distance / max_len), mengembalikan nilai dari 0 (tidak mirip) hingga 1 (identik), dengan penanganan kasus khusus seperti string kosong. Fungsi fuzzy_search melakukan pencarian fuzzy dengan mengubah teks dan pola ke huruf kecil, memecah teks menjadi kata-kata, lalu menghitung kemiripan setiap kata dengan pola menggunakan calculate_similarity. Jika kemiripan melebihi ambang batas (threshold), posisi awal kata dalam teks dan skor kemiripannya disimpan dalam daftar hasil, yang kemudian diurutkan berdasarkan skor secara menurun. Terakhir, fungsi fuzzy_search_all memperluas fungsionalitas ini untuk mencari beberapa pola dalam teks, dengan mengiterasi setiap pola, memanggil fuzzy_search, dan menyimpan hasil non-kosong dalam kamus dengan kunci pola dan nilai daftar pasangan (indeks, skor kemiripan). Alur ini memungkinkan pencarian teks yang fleksibel dan toleran terhadap variasi ejaan, dengan efisiensi yang dioptimalkan untuk kasus penggunaan seperti pencarian CV atau dokumen.

4.1.6. main.py

```
"""
Main entry point for CV Analyzer application.
"""
from gui.main_window import CVAnalyzerApp

def main():
    app = CVAnalyzerApp()
    app.run()

if __name__ == "__main__":
    main()
```

File ini berfungsi sebagai titik masuk utama (entry point) untuk aplikasi CV Analyzer. Kode ini mengimpor kelas CVAnalyzerApp dari modul gui.main_window, mendefinisikan fungsi main() yang membuat instance CVAnalyzerApp dan menjalankan aplikasi dengan memanggil metode run(), serta memastikan fungsi main() hanya dijalankan jika file dijalankan langsung (bukan diimpor sebagai modul) melalui pengecekan

if __name__ == "__main__". File ini bertugas memulai aplikasi GUI untuk analisis CV.

4.1.7. init_db.py

```
import os
import mysql.connector
from dotenv import load_dotenv, find_dotenv

load_dotenv(find_dotenv(), override=True)

def create_database():
    host     = os.getenv("DB_HOST")
    user     = os.getenv("DB_USER")
    pwd      = os.getenv("DB_PASSWORD")
    db_name  = os.getenv("DB_NAME")

    print(f"[DEBUG] create_database() menggunakan: host={host},
user={user}, pwd={'***' if pwd else None}")
    cnx = mysql.connector.connect(
        host=host,
        user=user,
        password=pwd
    )
    cur = cnx.cursor()
    cur.execute(
        f"CREATE DATABASE IF NOT EXISTS `{db_name}` "
        "CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;"
    )
    cnx.commit()
    cur.close()
    cnx.close()
    print(f"[OK] Database `{db_name}` siap (dibuat jika belum ada).")

def run_sql_file(path: str):
    create_database()

    # Create database config
    db_config = {
        'host': os.getenv('DB_HOST'),
        'user': os.getenv('DB_USER'),
        'password': os.getenv('DB_PASSWORD'),
        'database': os.getenv('DB_NAME')
    }

    # Import here to avoid circular import
    from database_connector.db import connect_db
    conn = connect_db(db_config)
    cursor = conn.cursor()
    print(f"[DEBUG] Menjalankan seeding SQL: {path}")
    with open(path, 'r', encoding='utf-8') as f:
        sql = f.read()

    for stmt in sql.split(';'):
        stmt = stmt.strip()
        if not stmt:
            continue
        try:
            cursor.execute(stmt)
        except mysql.connector.Error as err:
            print(f"[ERROR] Gagal eksekusi: {stmt[:80]}…\n   → {err}")

    conn.commit()
    cursor.close()
    conn.close()
    print("[OK] Tabel & data seed berhasil dibuat.")

if __name__ == "__main__":
    base_dir = os.path.dirname(__file__)
```

```
    sql_path = os.path.abspath(os.path.join(base_dir, "..", "data",
"tubes3_seeding.sql"))
    run_sql_file(sql_path)
```

Kelas init_db dirancang untuk menginisialisasi *database* MySQL yang dibutuhkan untuk aplikasi. Kelas ini digunakan untuk memudahkan agar tidak perlu membuat perintah SQL secara manual. Fungsi create_database() digunakan untuk membuat skema basis data yang nantinya akan digunakan untuk keperluan aplikasi. Fungsi run_sql_file(path: str) digunakan untuk *seeding* file SQL.

4.1.8. main_window.py

```
import tkinter as tk
from tkinter import filedialog, messagebox
import customtkinter as ctk
import os
import sys
import threading
import time
from typing import Dict, List, Tuple
from dotenv import load_dotenv
import subprocess
from database_connector.query_service import QueryService
from pattern.kmp import kmp_search
from pattern.bm import boyer_moore_search
from pattern.fuzzy import fuzzy_search_all

# Add parent directory to path to import from other folders
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file
__))))

from extract.cv_extractor import CVExtractor
from extract.regex import parse_cv_sections
from gui.summary_window import SummaryWindow

class CVAnalyzerApp:
    def __init__(self):
        load_dotenv()
        ctk.set_appearance_mode("dark")
        ctk.set_default_color_theme("blue")

        self.root = ctk.CTk()
        self.root.title("LinkedOut")
        self.root.geometry("1400x800")
        self.root.minsize(1200, 700)

        self.query_service = QueryService()
        self.current_file_paths: List[str] = []
        self.analysis_results: Dict[str, dict] = {}
        self.applicant_data: Dict[str, dict] = {}
        self.search_algorithm = "KMP"  # Default to KMP

        self.setup_ui()

    def setup_ui(self):
        # Main Scrollable Frame
        self.main_scrollable = ctk.CTkScrollableFrame(self.root,
corner_radius=0, fg_color="transparent")
        self.main_scrollable.pack(fill="both", expand=True)

        # Main Container
        main_frame = ctk.CTkFrame(self.main_scrollable,
corner_radius=0, fg_color="transparent")
        main_frame.pack(fill="both", expand=True, padx=30, pady=30)

        # Header section
        header_frame = ctk.CTkFrame(main_frame, height=100,
```

```
corner_radius=20, fg_color=("#2B2B2B", "#1A1A1A"))
        header_frame.pack(fill="x", pady=(0, 25))
        header_frame.pack_propagate(False)
        ctk.CTkLabel(header_frame, text="LinkedOut",
font=ctk.CTkFont(size=32, weight="bold", family="Helvetica"),
text_color="#4A90E2").place(relx=0.5, rely=0.35, anchor="center")
        ctk.CTkLabel(header_frame, text="Analyze Multiple CVs Easily",
font=ctk.CTkFont(size=14, family="Helvetica"),
text_color="#A0A0A0").place(relx=0.5, rely=0.75, anchor="center")

        # Search section
        search_frame = ctk.CTkFrame(main_frame, corner_radius=20,
fg_color=("#2B2B2B", "#1A1A1A"))
        search_frame.pack(fill="x", pady=(0, 25))
        search_row = ctk.CTkFrame(search_frame,
fg_color="transparent")
        search_row.pack(pady=30)

        # Search input
        self.search_entry = ctk.CTkEntry(
            search_row,
            placeholder_text="Enter keywords (e.g., React, Tailwind,
HTML)",
            font=ctk.CTkFont(size=14, family="Helvetica"),
            height=40,
            width=400
        )
        self.search_entry.pack(side="left", padx=(0, 10))

        # Number of matches input
        self.matches_entry = ctk.CTkEntry(
            search_row,
            placeholder_text="Number of matches",
            font=ctk.CTkFont(size=14, family="Helvetica"),
            height=40,
            width=150
        )
        self.matches_entry.pack(side="left", padx=(0, 10))
        self.matches_entry.insert(0, "5")  # Default value

        # Algorithm selection
        self.algorithm_var = ctk.StringVar(value="KMP")
        algorithm_frame = ctk.CTkFrame(search_row,
fg_color="transparent")
        algorithm_frame.pack(side="left", padx=(0, 10))

        ctk.CTkLabel(
            algorithm_frame,
            text="Algorithm:",
            font=ctk.CTkFont(size=14, family="Helvetica")
        ).pack(side="left", padx=(0, 5))

        # Replace radio buttons with segmented button
        self.algorithm_selector = ctk.CTkSegmentedButton(
            algorithm_frame,
            values=["KMP", "BM"],
            variable=self.algorithm_var,
            font=ctk.CTkFont(size=14, family="Helvetica"),
            height=35,
            width=200,
            fg_color="#2B2B2B",
            selected_color="#4A90E2",
            selected_hover_color="#357ABD",
            unselected_color="#1A1A1A",
            unselected_hover_color="#2B2B2B",
            command=self.update_algorithm_label
        )
        self.algorithm_selector.pack(side="left", padx=(0, 10))

        # Add label to show current algorithm
        self.algorithm_label = ctk.CTkLabel(
            algorithm_frame,
            text="(Knuth-Morris-Pratt)",
            font=ctk.CTkFont(size=12, family="Helvetica"),
            text_color="#A0A0A0"
```

```
        )
        self.algorithm_label.pack(side="left")

        # Search button
        self.search_btn = ctk.CTkButton(
            search_row,
            text="Search",
            font=ctk.CTkFont(size=14, weight="bold",
family="Helvetica"),
            height=40,
            width=100,
            fg_color="#4A90E2",
            hover_color="#357ABD",
            command=self.search_keywords
        )
        self.search_btn.pack(side="left", padx=(0, 20))

        # Upload section
        upload_frame = ctk.CTkFrame(main_frame, corner_radius=20,
fg_color=("#2B2B2B", "#1A1A1A"))
        upload_frame.pack(fill="x", pady=(0, 25))
        self.upload_btn = ctk.CTkButton(
            upload_frame,
            text="Upload CVs (PDF)",
            font=ctk.CTkFont(size=16, weight="bold",
family="Helvetica"),
            height=50,
            corner_radius=25,
            fg_color="#4A90E2",
            hover_color="#357ABD",
            command=self.upload_files
        )
        self.upload_btn.pack(pady=20)
        self.file_info_label = ctk.CTkLabel(
            upload_frame,
            text="No files selected",
            font=ctk.CTkFont(size=12, family="Helvetica"),
            text_color="#A0A0A0"
        )
        self.file_info_label.pack(pady=(0, 20))

        # Results section
        self.results_frame = ctk.CTkFrame(main_frame,
corner_radius=20, fg_color=("#2B2B2B", "#1A1A1A"))
        self.results_frame.pack(fill="both", expand=True)

        # Initial message
        self.show_initial_message()

    def show_initial_message(self):
        message_label = ctk.CTkLabel(
            self.results_frame,
            text="Upload CVs and search for matches to see results",
            font=ctk.CTkFont(size=16),
            text_color="#A0A0A0"
        )
        message_label.place(relx=0.5, rely=0.5, anchor="center")

    def upload_files(self):
        file_paths = filedialog.askopenfilenames(title="Select CV PDF
Files", filetypes=[("PDF files", "*.pdf")])
        if file_paths:
            self.current_file_paths = list(file_paths)
            self.file_info_label.configure(text=f"Selected:
{len(file_paths)} files")
            self.upload_btn.configure(text="Analyzing...",
state="disabled", fg_color="#666666")

            # Start analysis in background
            threading.Thread(target=self.analyze_cvs,
daemon=True).start()

    def analyze_cvs(self):
        try:
            for file_path in self.current_file_paths:
```

```
                print(f"\nProcessing file: {file_path}")

                # Extract text using CVExtractor
                cv_extractor = CVExtractor(file_path)
                cv_extractor.process()
                raw_text = cv_extractor.retrieve_raw_text()
                print("\nExtracted text sample:")
                print(raw_text[:500] + "...")

                # Extract sections using regex
                sections = parse_cv_sections(raw_text)

                # Store results
                results = {
                    'summary': sections['summary'],
                    'skills': sections['skills'],
                    'experience': sections['experience'],
                    'education': sections['education'],
                    'text': raw_text  # Keep raw text for keyword
search
                }
                self.analysis_results[file_path] = results

                # Get applicant data from database
                try:
                    applicant_data =
self.query_service.get_applicant_by_cv_path(file_path)
                    if applicant_data:
                        self.applicant_data[file_path] =
applicant_data
                    else:
                        # If no applicant found, use default values
                        self.applicant_data[file_path] = {
                            'first_name': 'Unknown',
                            'last_name': 'Unknown',
                            'date_of_birth': 'Unknown',
                            'address': 'Unknown',
                            'phone_number': 'Unknown',
                            'cv_path': file_path
                        }
                except Exception as e:
                    print(f"Error getting applicant data: {str(e)}")
                    # Use default values on error
                    self.applicant_data[file_path] = {
                        'first_name': 'Unknown',
                        'last_name': 'Unknown',
                        'date_of_birth': 'Unknown',
                        'address': 'Unknown',
                        'phone_number': 'Unknown',
                        'cv_path': file_path
                    }

                self.root.after(0, self.update_file_status, file_path,
"✓")
        except Exception as e:
            print(f"Error in analyze_cvs: {str(e)}")
            self.root.after(0, self.show_error, str(e))
        finally:
            self.root.after(0, self.reset_upload_button)

    def update_file_status(self, file_path: str, status: str):
        """Update the status of a file in the list"""
        self.file_info_label.configure(text=f"Processed:
{len(self.analysis_results)}/{len(self.current_file_paths)} files")

    def reset_upload_button(self):
        self.upload_btn.configure(text="Upload CVs (PDF)",
state="normal", fg_color="#4A90E2")

    def show_error(self, error_message):
        messagebox.showerror("Analysis Error", f"An error occurred:
{error_message}")

    def calculate_match_score(self, text: str, keywords: List[str]) ->
Tuple[float, Dict[str, int], float, float]:
```

```
        """Calculate match score using selected pattern matching
algorithm and fuzzy matching
        Returns: (total_score, keyword_matches, exact_time,
fuzzy_time)"""
        text = text.lower()
        total_score = 0
        keyword_matches = {k.lower(): 0 for k in keywords}  # Track
matches per keyword
        exact_matches = set()  # Track which keywords had exact
matches

        # Get selected algorithm
        selected_algo = self.algorithm_var.get()
        print(f"Using algorithm: {selected_algo}")

        # Measure exact matching time
        exact_start_time = time.time()

        # First try exact matching
        for keyword in keywords:
            keyword = keyword.lower()
            # Use selected algorithm for pattern matching
            if selected_algo == "KMP":
                matches = kmp_search(text, keyword)
            else:  # Boyer-Moore
                matches = boyer_moore_search(text, keyword)

            if matches:  # If exact matches found
                exact_matches.add(keyword)
                keyword_matches[keyword] = len(matches)
                print(f"Found {len(matches)} matches for '{keyword}'
using {selected_algo}")
                # Calculate score based on number of matches and
context
                for match_pos in matches:
                    # Get context around the match
                    context_start = max(0, match_pos - 50)
                    context_end = min(len(text), match_pos +
len(keyword) + 50)
                    context = text[context_start:context_end]

                    # Calculate score based on context length
                    score = 1.0 / (1.0 + len(context))
                    total_score += score

        exact_time = time.time() - exact_start_time

        # Measure fuzzy matching time
        fuzzy_time = 0
        unmatched_keywords = [k for k in keywords if k not in
exact_matches]
        if unmatched_keywords:
            print(f"Trying fuzzy matching for: {unmatched_keywords}")
            fuzzy_start_time = time.time()
            fuzzy_results = fuzzy_search_all(text, unmatched_keywords,
threshold=0.8)
            for keyword, matches in fuzzy_results.items():
                keyword_matches[keyword] = len(matches)
                for match_pos, similarity in matches:
                    # Get context around the match
                    context_start = max(0, match_pos - 50)
                    context_end = min(len(text), match_pos +
len(keyword) + 50)
                    context = text[context_start:context_end]

                    # Calculate score based on context length and
similarity
                    # Use similarity score directly from fuzzy search
                    score = (similarity * 0.5) / (1.0 + len(context))
# Reduce weight of fuzzy matches
                    total_score += score
            fuzzy_time = time.time() - fuzzy_start_time

        return total_score, keyword_matches, exact_time, fuzzy_time
```

```
    def search_keywords(self):
        if not self.current_file_paths:
            messagebox.showwarning("Warning", "Please upload and
analyze CVs first.")
            return

        keywords_input = self.search_entry.get().strip()
        if not keywords_input:
            messagebox.showwarning("Warning", "Please enter keywords
to search.")
            return

        try:
            num_matches = int(self.matches_entry.get())
            if num_matches <= 0:
                raise ValueError("Number of matches must be positive")
        except ValueError:
            messagebox.showwarning("Warning", "Please enter a valid
number of matches.")
            return

        keywords = [k.strip() for k in keywords_input.split(',') if
k.strip()]

        # Calculate match scores for each file
        file_scores = []
        total_exact_time = 0
        total_fuzzy_time = 0

        for file_path in self.current_file_paths:
            if file_path in self.analysis_results:
                results = self.analysis_results[file_path]
                if 'text' in results:
                    score, keyword_matches, exact_time, fuzzy_time =
self.calculate_match_score(results['text'], keywords)
                    total_exact_time += exact_time
                    total_fuzzy_time += fuzzy_time
                    if score > 0:
                        file_scores.append((file_path, score,
keyword_matches))

        # Sort by score and take top matches
        file_scores.sort(key=lambda x: x[1], reverse=True)
        top_matches = file_scores[:num_matches]

        # Clear previous results
        for widget in self.results_frame.winfo_children():
            widget.destroy()

        if top_matches:
            # Create scrollable frame for cards
            cards_frame = ctk.CTkScrollableFrame(self.results_frame,
fg_color="transparent")
            cards_frame.pack(fill="both", expand=True, padx=20,
pady=20)

            # Add timing information
            timing_frame = ctk.CTkFrame(cards_frame,
fg_color="transparent")
            timing_frame.pack(fill="x", pady=(0, 10))

            # Exact matching time
            exact_time_label = ctk.CTkLabel(
                timing_frame,
                text=f"Exact Matching Time
({self.algorithm_var.get()}): {total_exact_time:.3f} seconds",
                font=ctk.CTkFont(size=12, family="Helvetica"),
                text_color="#4A90E2"
            )
            exact_time_label.pack(side="left", padx=(0, 20))

            # Fuzzy matching time
            if total_fuzzy_time > 0:
                fuzzy_time_label = ctk.CTkLabel(
                    timing_frame,
```

```
                    text=f"Fuzzy Matching Time: {total_fuzzy_time:.3f}
seconds",
                    font=ctk.CTkFont(size=12, family="Helvetica"),
                    text_color="#4A90E2"
                )
                fuzzy_time_label.pack(side="left")

            # Create cards for each match
            for file_path, score, keyword_matches in top_matches:
                card = self.create_match_card(cards_frame, file_path,
score, keyword_matches)
                card.pack(fill="x", pady=10)
        else:
            # Show no matches message
            message_label = ctk.CTkLabel(
                self.results_frame,
                text="No matches found",
                font=ctk.CTkFont(size=16),
                text_color="#A0A0A0"
            )
            message_label.place(relx=0.5, rely=0.5, anchor="center")

    def create_match_card(self, parent, file_path: str, score: float,
keyword_matches: Dict[str, int]) -> ctk.CTkFrame:
        """Create a card for displaying a match"""
        card = ctk.CTkFrame(parent, corner_radius=10,
fg_color=("#2B2B2B", "#1A1A1A"))

        # Get applicant data
        applicant_data = self.applicant_data.get(file_path, {})
        name = f"{applicant_data.get('first_name', 'Unknown')}
{applicant_data.get('last_name', 'Unknown')}"

        # Card content
        content_frame = ctk.CTkFrame(card, fg_color="transparent")
        content_frame.pack(fill="both", expand=True, padx=20, pady=20)

        # Name and score
        header_frame = ctk.CTkFrame(content_frame,
fg_color="transparent")
        header_frame.pack(fill="x", pady=(0, 10))

        name_label = ctk.CTkLabel(
            header_frame,
            text=name,
            font=ctk.CTkFont(size=18, weight="bold")
        )
        name_label.pack(side="left")

        # Score and matches info
        info_frame = ctk.CTkFrame(header_frame,
fg_color="transparent")
        info_frame.pack(side="right")

        score_label = ctk.CTkLabel(
            info_frame,
            text=f"Match Score: {score:.2f}",
            font=ctk.CTkFont(size=14),
            text_color="#4A90E2"
        )
        score_label.pack(side="top", pady=(0, 5))

        # Total matches
        total_matches = sum(keyword_matches.values())
        matches_label = ctk.CTkLabel(
            info_frame,
            text=f"Total Matches: {total_matches}",
            font=ctk.CTkFont(size=14),
            text_color="#4A90E2"
        )
        matches_label.pack(side="top", pady=(0, 5))

        # Detailed matches per keyword
        matches_detail = ctk.CTkFrame(content_frame,
fg_color="transparent")
```

```
        matches_detail.pack(fill="x", pady=(0, 10))

        ctk.CTkLabel(
            matches_detail,
            text="Matches per keyword:",
            font=ctk.CTkFont(size=12, weight="bold"),
            text_color="#A0A0A0"
        ).pack(anchor="w", pady=(0, 5))

        for keyword, count in keyword_matches.items():
            if count > 0:  # Only show keywords that have matches
                match_row = ctk.CTkFrame(matches_detail,
fg_color="transparent")
                match_row.pack(fill="x", pady=2)

                ctk.CTkLabel(
                    match_row,
                    text=f"• {keyword.capitalize()}:",
                    font=ctk.CTkFont(size=12),
                    text_color="#A0A0A0"
                ).pack(side="left")

                ctk.CTkLabel(
                    match_row,
                    text=f"{count} matches",
                    font=ctk.CTkFont(size=12),
                    text_color="#4A90E2"
                ).pack(side="left", padx=(5, 0))

        # Buttons
        buttons_frame = ctk.CTkFrame(content_frame,
fg_color="transparent")
        buttons_frame.pack(fill="x", pady=(10, 0))

        view_summary_btn = ctk.CTkButton(
            buttons_frame,
            text="View Summary",
            font=ctk.CTkFont(size=14, weight="bold"),
            height=35,
            width=120,
            fg_color="#4A90E2",
            hover_color="#357ABD",
            command=lambda: self.show_summary(file_path)
        )
        view_summary_btn.pack(side="left", padx=(0, 10))

        view_cv_btn = ctk.CTkButton(
            buttons_frame,
            text="View CV",
            font=ctk.CTkFont(size=14, weight="bold"),
            height=35,
            width=120,
            fg_color="#2B2B2B",
            hover_color="#1A1A1A",
            command=lambda: self.open_cv(file_path)
        )
        view_cv_btn.pack(side="left")

        return card

    def show_summary(self, file_path: str):
        """Show summary window for a CV"""
        if file_path in self.analysis_results and file_path in
self.applicant_data:
            SummaryWindow(
                self.root,
                self.analysis_results[file_path],
                self.applicant_data[file_path],
                file_path
            )

    def open_cv(self, file_path: str):
        """Open CV file"""
        try:
            if os.path.exists(file_path):
```

```
                if os.name == 'nt':  # Windows
                    os.startfile(file_path)
                else:  # Linux/Mac
                    subprocess.run(['xdg-open', file_path])
            else:
                messagebox.showerror("Error", "CV file not found")
        except Exception as e:
            messagebox.showerror("Error", f"Could not open CV:
{str(e)}")

    def update_algorithm_label(self, value):
        """Update the algorithm description label"""
        if value == "KMP":

self.algorithm_label.configure(text="(Knuth-Morris-Pratt)")
        else:
            self.algorithm_label.configure(text="(Boyer-Moore)")

    def run(self):
        self.root.mainloop()
```

File main_window.py berfungsi sebagai komponen utama GUI aplikasi untuk menganalisis CV berformat PDF. Kode ini mendefinisikan kelas CVAnalyzerApp yang mengatur antarmuka pengguna dengan customtkinter, termasuk frame untuk header, kolom pencarian, tombol unggah, dan area hasil. Fungsi utamanya meliputi inisialisasi aplikasi, pengaturan tema gelap, pembuatan elemen UI seperti kolom input kata kunci, pemilih algoritma (KMP/Boyer-Moore), tombol pencarian, dan tombol unggah file PDF. File ini juga menangani logika pengunggahan file PDF, memicu analisis CV di latar belakang menggunakan CVExtractor dan parse_cv_sections, menyimpan hasil analisis dan data pelamar, serta melakukan pencarian kata kunci dengan algoritma yang dipilih dan pencarian fuzzy. Selain itu, kode ini menampilkan hasil pencarian dalam bentuk kartu dengan skor kecocokan dan detail pelamar, menyediakan tombol untuk melihat ringkasan CV atau membuka file PDF, serta mengukur waktu eksekusi pencocokan eksak dan fuzzy. File ini juga mengelola pembaruan status UI, menangani error, dan menjalankan loop utama aplikasi.

4.1.9. summary_window.py

```
import tkinter as tk
from tkinter import messagebox
import customtkinter as ctk
import os
import subprocess
from typing import Dict

class SummaryWindow:
    def __init__(self, parent, cv_data: Dict, applicant_data: Dict,
file_path: str):
        self.window = ctk.CTkToplevel(parent)
        self.window.title("CV Summary")
        self.window.geometry("1200x800")
        self.window.minsize(1000, 600)

        self.cv_data = cv_data
        self.applicant_data = applicant_data
        self.file_path = file_path

        self.setup_ui()

    def setup_ui(self):
```

```
        # Main container
        main_frame = ctk.CTkFrame(self.window, corner_radius=0,
fg_color="transparent")
        main_frame.pack(fill="both", expand=True, padx=30, pady=30)

        # Header with applicant info
        header_frame = ctk.CTkFrame(main_frame, height=150,
corner_radius=20, fg_color=("#2B2B2B", "#1A1A1A"))
        header_frame.pack(fill="x", pady=(0, 25))
        header_frame.pack_propagate(False)

        # Applicant info
        info_frame = ctk.CTkFrame(header_frame,
fg_color="transparent")
        info_frame.pack(fill="both", expand=True, padx=20, pady=20)

        # Name
        name_label = ctk.CTkLabel(
            info_frame,
            text=f"{self.applicant_data['first_name']}
{self.applicant_data['last_name']}",
            font=ctk.CTkFont(size=24, weight="bold")
        )
        name_label.pack(anchor="w", pady=(0, 10))

        # Other details in a grid
        details_frame = ctk.CTkFrame(info_frame,
fg_color="transparent")
        details_frame.pack(fill="x")

        # Date of Birth
        dob_label = ctk.CTkLabel(
            details_frame,
            text=f"Date of Birth:
{self.applicant_data['date_of_birth']}",
            font=ctk.CTkFont(size=14)
        )
        dob_label.grid(row=0, column=0, padx=(0, 20), pady=5,
sticky="w")

        # Phone
        phone_label = ctk.CTkLabel(
            details_frame,
            text=f"Phone: {self.applicant_data['phone_number']}",
            font=ctk.CTkFont(size=14)
        )
        phone_label.grid(row=0, column=1, padx=(0, 20), pady=5,
sticky="w")

        # Address
        address_label = ctk.CTkLabel(
            details_frame,
            text=f"Address: {self.applicant_data['address']}",
            font=ctk.CTkFont(size=14)
        )
        address_label.grid(row=1, column=0, columnspan=2, pady=5,
sticky="w")

        # View CV button
        view_cv_btn = ctk.CTkButton(
            header_frame,
            text="View CV",
            font=ctk.CTkFont(size=14, weight="bold"),
            height=40,
            width=120,
            fg_color="#4A90E2",
            hover_color="#357ABD",
            command=self.open_cv
        )
        view_cv_btn.place(relx=0.95, rely=0.5, anchor="e")

        # Content sections
        content_frame = ctk.CTkFrame(main_frame, corner_radius=20,
fg_color=("#2B2B2B", "#1A1A1A"))
        content_frame.pack(fill="both", expand=True)
```

```
        # Create columns for different sections
        columns = [
            ("Summary", "summary"),
            ("Skills", "skills"),
            ("Experience", "experience"),
            ("Education", "education")
        ]

        self.info_textboxes = {}
        for title, key in columns:
            column_frame = ctk.CTkFrame(content_frame,
fg_color="transparent")
            column_frame.pack(side="left", fill="both", expand=True,
padx=5, pady=5)

            ctk.CTkLabel(
                column_frame,
                text=title,
                font=ctk.CTkFont(size=16, weight="bold")
            ).pack(pady=(0, 10))

            textbox = ctk.CTkTextbox(
                column_frame,
                wrap="word",
                font=ctk.CTkFont(size=12, family="Helvetica")
            )
            textbox.pack(fill="both", expand=True)
            self.info_textboxes[key] = textbox

        self.display_results()

    def display_results(self):
        for key in self.info_textboxes:
            self.info_textboxes[key].configure(state="normal")
            self.info_textboxes[key].delete("1.0", "end")
            self.info_textboxes[key].insert("1.0",
self.cv_data.get(key, 'N/A'))
            self.info_textboxes[key].configure(state="disabled")

    def open_cv(self):
        """Open CV file"""
        try:
            if os.path.exists(self.file_path):
                if os.name == 'nt':  # Windows
                    os.startfile(self.file_path)
                else:  # Linux/Mac
                    subprocess.run(['xdg-open', self.file_path])
            else:
                messagebox.showerror("Error", "CV file not found")
        except Exception as e:
            messagebox.showerror("Error", f"Could not open CV:
{str(e)}")
```

File summary_window.py berfungsi untuk membuat jendela sekunder (Toplevel) dalam aplikasi GUI yang menampilkan ringkasan data CV dan informasi pelamar. Kode ini mendefinisikan kelas SummaryWindow yang menginisialisasi jendela dengan antarmuka menggunakan customtkinter, menampilkan informasi pelamar seperti nama, tanggal lahir, nomor telepon, dan alamat dari applicant_data, serta menampilkan bagian CV seperti ringkasan, keterampilan, pengalaman, dan pendidikan dari cv_data. File ini mengatur tata letak UI dengan frame utama, header untuk informasi pelamar, tombol untuk membuka file PDF CV, dan empat kolom textbox untuk menampilkan data CV secara terorganisir. Fungsi display_results mengisi textbox dengan data CV yang sesuai, sedangkan fungsi open_cv memungkinkan pengguna membuka file PDF

menggunakan aplikasi default sistem operasi, dengan penanganan error jika file tidak ditemukan atau gagal dibuka.

4.1.10. db.py

```
import os
from dotenv import load_dotenv, find_dotenv
import mysql.connector
from mysql.connector import Error
from typing import Dict, Any, Optional

load_dotenv(find_dotenv(), override=True)

def connect_db(config: Dict[str, str] = None) ->
Optional[mysql.connector.MySQLConnection]:
    """
    Create a database connection using the provided configuration.

    Args:
        config (Dict[str, str], optional): Database configuration
dictionary containing
            host, user, password, and database name. If None, will use
environment variables.

    Returns:
        Optional[mysql.connector.MySQLConnection]: Database connection
if successful,
            None otherwise.
    """
    try:
        if config is None:
            config = {
                'host': os.getenv('DB_HOST'),
                'user': os.getenv('DB_USER'),
                'password': os.getenv('DB_PASSWORD'),
                'database': os.getenv('DB_NAME')
            }

        connection = mysql.connector.connect(
            host=config['host'],
            user=config['user'],
            password=config['password'],
            database=config['database']
        )
        if connection.is_connected():
            return connection
    except Error as e:
        print(f"Error connecting to MySQL database: {e}")
    return None

def execute_query(connection: mysql.connector.MySQLConnection, query:
str, params: tuple = None) -> Optional[Any]:
    """
    Execute a database query.

    Args:
        connection (mysql.connector.MySQLConnection): Database
connection
        query (str): SQL query to execute
        params (tuple, optional): Query parameters

    Returns:
        Optional[Any]: Query result if successful, None otherwise
    """
    try:
        cursor = connection.cursor()
        if params:
            cursor.execute(query, params)
        else:
            cursor.execute(query)

        if query.strip().upper().startswith('SELECT'):
```

```
            result = cursor.fetchall()
        else:
            connection.commit()
            result = cursor.rowcount

        cursor.close()
        return result
    except Error as e:
        print(f"Error executing query: {e}")
        return None
```

File ini bertanggung jawab untuk mengelola koneksi dan eksekusi query ke database MySQL. Fungsi connect_db membuat koneksi ke database menggunakan konfigurasi yang diberikan atau variabel lingkungan dari file .env (seperti host, user, password, dan nama database), lalu mengembalikan objek koneksi jika berhasil atau None jika gagal. Fungsi execute_query menjalankan query SQL pada koneksi yang diberikan, mendukung query dengan parameter opsional, dan mengembalikan hasil untuk query SELECT (data yang diambil) atau jumlah baris yang terpengaruh untuk query non-SELECT (seperti INSERT atau UPDATE), dengan penanganan error untuk memastikan eksekusi yang aman. File ini menyediakan fungsi dasar untuk interaksi database dalam aplikasi, seperti mengambil data pelamar untuk analisis CV.

4.1.11. query_service.py

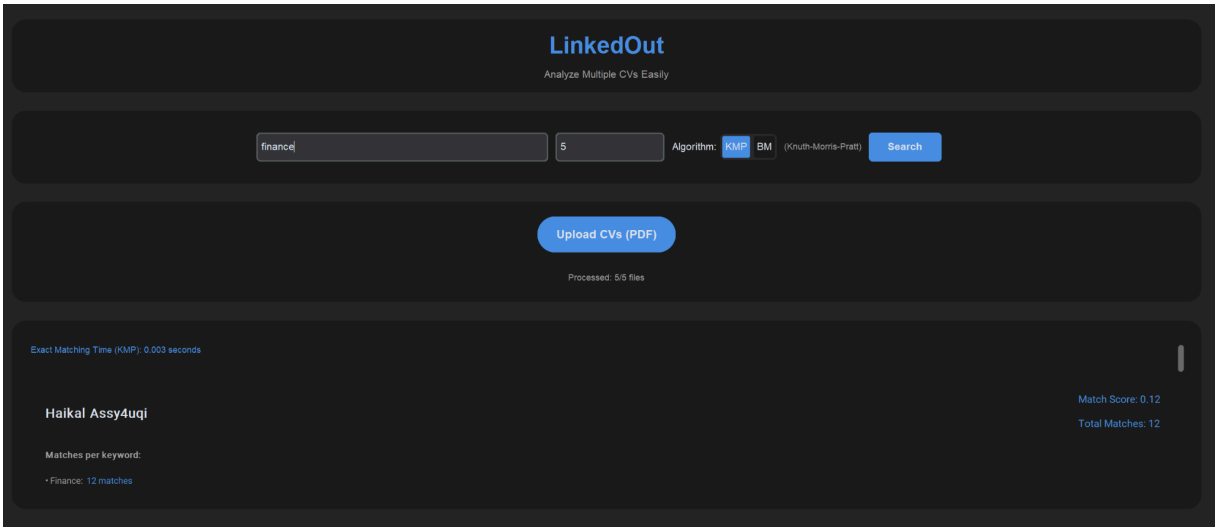
```
import os
from typing import Dict, List, Optional
from dotenv import load_dotenv
from .db import connect_db, execute_query

# Load environment variables
load_dotenv()

class QueryService:
    def __init__(self):
        self.db_config = {
            'host': os.getenv('DB_HOST', 'localhost'),
            'user': os.getenv('DB_USER', 'root'),
            'password': os.getenv('DB_PASSWORD', ''),
            'database': os.getenv('DB_NAME', 'cv_analyzer')
        }

    def get_applicant_by_cv_path(self, cv_path: str) ->
Optional[Dict]:
        """
        Get applicant information by CV path.
        Returns None if no applicant is found with the given CV path.
        """
        conn = connect_db()
        cursor = conn.cursor(dictionary=True)

        # Extract just the data/... part from the full path
        # Example: from
"C:/Users/USER/Downloads/archive/data/data/CHEF/10001727.pdf"
        # to "data/CHEF/10001727.pdf"
        try:
            # Find the last occurrence of "data/" in the path
            data_index = cv_path.rindex("data/")
            db_path = cv_path[data_index:]  # Get everything from
"data/" onwards
        except ValueError:
            # If "data/" not found, use the original path
            db_path = cv_path
```

```
        print(f"Looking up path in database: {db_path}")  # Debug
print

        sql = """
            SELECT ap.*
            FROM ApplicantProfile ap
            JOIN ApplicationDetail ad ON ap.applicant_id =
ad.applicant_id
            WHERE ad.cv_path = %s
        """
        cursor.execute(sql, (db_path,))
        row = cursor.fetchone()

        cursor.close()
        conn.close()
        return row

def insert_applicant(first_name: str,
                     last_name: str,
                     dob: str,
                     address: str,
                     phone: str) -> int:
    """
    Masukkan data ApplicantProfile baru,
    return applicant_id (auto increment).
    """
    conn = connect_db()
    cursor = conn.cursor()

    sql = """
        INSERT INTO ApplicantProfile
            (first_name, last_name, date_of_birth, address,
phone_number)
        VALUES (%s, %s, %s, %s, %s)
    """
    cursor.execute(sql, (first_name, last_name, dob, address, phone))
    new_id = cursor.lastrowid

    conn.commit()
    cursor.close()
    conn.close()
    return new_id

def insert_application_detail(applicant_id: int,
                              role: str,
                              cv_path: str) -> None:
    """
    Masukkan data ApplicationDetail untuk applicant_id tertentu.
    """
    conn = connect_db()
    cursor = conn.cursor()

    sql = """
        INSERT INTO ApplicationDetail
            (applicant_id, application_role, cv_path)
        VALUES (%s, %s, %s)
    """
    cursor.execute(sql, (applicant_id, role, cv_path))

    conn.commit()
    cursor.close()
    conn.close()

def get_all_applicants() -> list[dict]:
    """
    Ambil semua baris dari ApplicantProfile
    sebagai list of dict.
    """
    conn = connect_db()
    cursor = conn.cursor(dictionary=True)

    cursor.execute("SELECT * FROM ApplicantProfile")
    rows = cursor.fetchall()
```

```
    cursor.close()
    conn.close()
    return rows

def get_application_detail(applicant_id: int) -> list[dict]:
    """
    Ambil semua ApplicationDetail untuk applicant_id tertentu.
    """
    conn = connect_db()
    cursor = conn.cursor(dictionary=True)

    cursor.execute("""
        SELECT *
          FROM ApplicationDetail
         WHERE applicant_id = %s
    """, (applicant_id,))
    rows = cursor.fetchall()

    cursor.close()
    conn.close()
    return rows
```

File ini berfungsi sebagai lapisan layanan untuk mengelola interaksi dengan database MySQL dalam aplikasi analisis CV. Kelas QueryService menginisialisasi konfigurasi database dari variabel lingkungan atau nilai default. Fungsi get_applicant_by_cv_path mengambil informasi pelamar dari tabel ApplicantProfile berdasarkan jalur file CV, menormalkan jalur dengan memotong hingga bagian "data/" untuk pencocokan dengan database, dan mengembalikan data pelamar sebagai dictionary atau None jika tidak ditemukan. Fungsi insert_applicant memasukkan data pelamar baru ke tabel ApplicantProfile dan mengembalikan ID yang dihasilkan. Fungsi insert_application_detail menambahkan detail aplikasi, seperti peran dan jalur CV, ke tabel ApplicationDetail untuk pelamar tertentu. Fungsi get_all_applicants mengambil semua data pelamar dari ApplicantProfile sebagai daftar dictionary, dan get_application_detail mengambil semua detail aplikasi untuk pelamar tertentu berdasarkan applicant_id. File ini menyediakan fungsi untuk operasi CRUD (Create, Read) pada data pelamar dan aplikasi, mendukung pengelolaan data dalam aplikasi analisis CV.

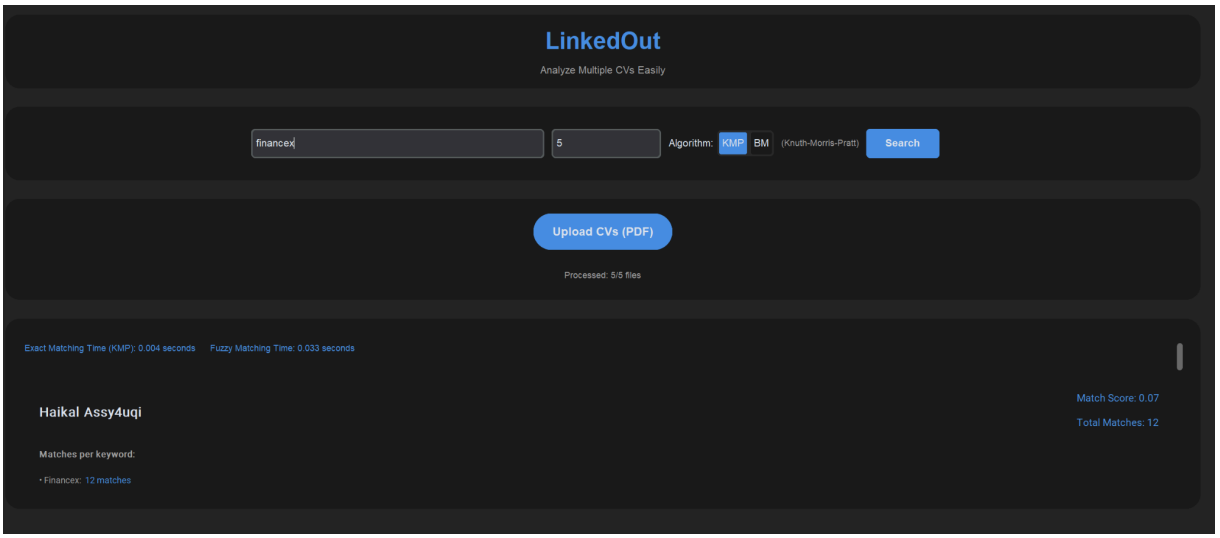
4.2. Pengujian dan Analisis



Gambar 4.2.1. Hasil pengujian Variasi Keyword 1 "Finance" Algoritma KMP

Rincian output dari 5 file pdf yang digunakan ialah
Exact Matching Time (KMP): 0.003 seconds

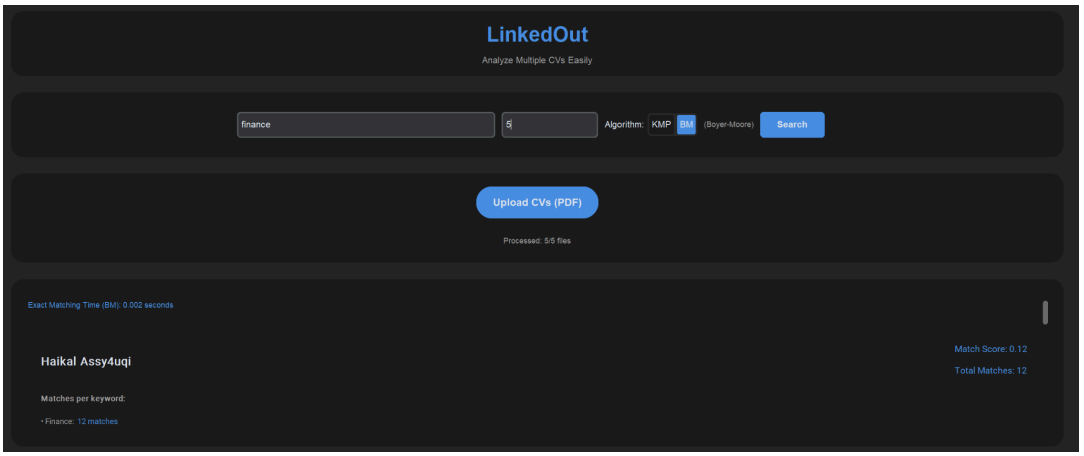
1. Haikal Assy4uqi
   Match Score : 0.12
   Total Matches : 2
   Matches per keyword : Finance 2 matches
2. AhM4d R4fi
   Match Score : 0.06
   Total Matches : 2
   Matches per keyword : Finance 2 matches
3. m0H4mmad nug9aha
   Match Score : 0.04
   Total Matches : 3
   Matches per keyword : Finance 2 matches
4. Unknown Unknown
   Match Score : 0.03
   Total Matches : 2
   Matches per keyword : Finance 2 matches
5. MOH4MM4D NUGR4H4
   Match Score : 0.03
   Total Matches : 2
   Matches per keyword : Finance 2 matches



Gambar 4.2.2. Hasil pengujian Variasi Keyword 2 "Finance" Algoritma KMP

Exact Matching Time (KMP): 0.004 seconds
Fuzzy Matching Time: 0.033 seconds

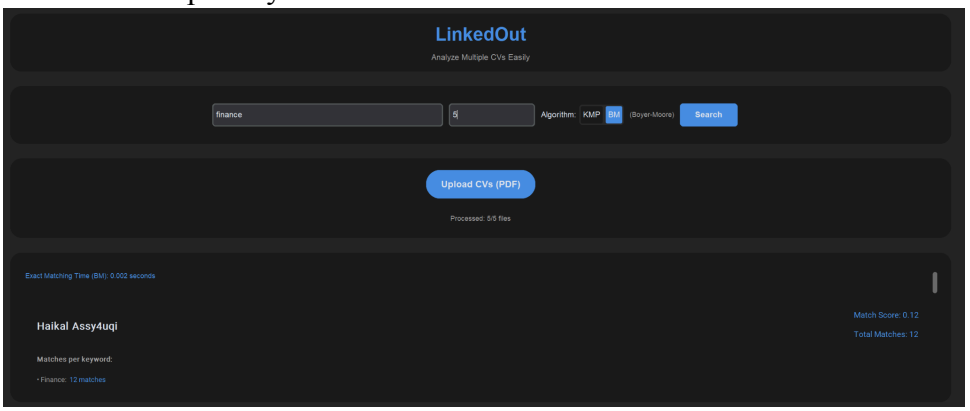
1. Haikal Assy4uqi
   Match Score : 0.07
   Total Matches : 12
   Matches per keyword : Financex 2 matches
2. AhM4d R4fi
   Match Score : 0.03
   Total Matches : 5
   Matches per keyword : Financex 2 matches
3. m0H4mmad nug9aha
   Match Score : 0.02
   Total Matches : 3
   Matches per keyword : Financex 2 matches
4. Unknown Unknown
   Match Score : 0.01
   Total Matches : 2
   Matches per keyword : Financex 2 matches
5. MOH4MM4D NUGR4H4
   Match Score : 0.01
   Total Matches : 2
   Matches per keyword : Financex 2 matches



Gambar 4.2.3. Hasil pengujian Variasi Algoritma dengan Keyword "finance" Algoritma BM

Exact Matching Time (KMP): 0.001 seconds

1. Haikal Assy4uqi
   Match Score : 0.12
   Total Matches : 12
   Matches per keyword : Finance 2 matches
2. AhM4d R4fi
   Match Score : 0.06
   Total Matches : 6
   Matches per keyword : Finance 2 matches
3. m0H4mmad nug9aha
   Match Score : 0.04
   Total Matches : 3
   Matches per keyword : Finance 2 matches
4. Unknown Unknown
   Match Score : 0.03
   Total Matches : 2
   Matches per keyword : Finance 2 matches
5. MOH4MM4D NUGR4H4
   Match Score : 0.03
   Total Matches : 2
   Matches per keyword : Finance 2 matches



Gambar 4.2.3. Hasil pengujian Variasi Panjang Keyword "finance manager"

Exact Matching Time (KMP): 0.003 seconds
Fuzzy Matching Time: 0.019 seconds
1. Unknown Unknown
   Match Score : 0.02
   Total Matches : 2
   Matches per keyword : Finance manager 2 matches
2. MOH4MM4D NUGR4H4
   Match Score : 0.02
   Total Matches : 2
   Matches per keyword : Finance manager 2 matches
3. m0H4mmad nug9aha
   Match Score : 0.02
   Total Matches : 2
   Matches per keyword : Finance manager 2 matches

Berikut adalah hasil analisis setiap alternatif solusi:

i. Variasi Keyword

Digunakan keyword "finance" lalu hasil program menampilkan exact match dari cv yang di-upload. Namun, ketika menggunakan pendekatan untuk merubah keyword menjadi 'typo' seperti "financex" maka akan melakukan pendekatan sebuah pattern yang paling mirip dengan kata "financex" yakni "finance". Selain itu, program juga dapat menerima multiple keywords dengan "," sebagai separator-nya.

ii. Variasi Algoritma yang Digunakan

Dengan menggunakan variasi algoritma baik KMP maupun BM cara bekerja algoritma menyesuaikan dengan keyword yang dimasukkan, baik dari segi runtime apabila keyword memiliki pola yang dapat teridentifikasi dalam sebuah runtime, maka algoritma KMP lebih baik menelurusinya namun apabila BM dia lebih bersifat general dan overral lebih cepat dibandingkan KMP. Fuzzy sendiri digunakan sebagai pattern untuk melakukan pendekatan pattern yang paling dekat dari hasil typing yang salah atau "typo" tersebut.

iii. Variasi Panjang Keyword

Panjang keyword berpengaruh signifikan terhadap kinerja algoritma pencocokan pola. Pada algoritma seperti Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM), keyword yang lebih panjang justru meningkatkan efisiensi karena struktur data pendukung (seperti prefix table atau heuristic shifting) menjadi lebih efektif dalam menghindari pencarian ulang di teks. Boyer-Moore, khususnya, mampu melakukan lompatan lebih jauh dalam teks saat keyword panjang, membuatnya sangat cepat dalam praktik. Sebaliknya, pada algoritma berbasis Levenshtein atau fuzzy matching, panjang

keyword justru menambah kompleksitas komputasi secara signifikan karena perhitungan jarak edit meningkat kuadrat terhadap panjang keyword dan teks, sehingga kurang efisien untuk pattern yang panjang, terutama dalam skenario real-time atau big data.

4. Kesimpulan dan Saran

a. Kesimpulan

Aplikasi pencarian data pelamar berbasis CV telah berhasil dibangun sesuai dengan spesifikasi yang ditentukan. Sistem ini mampu melakukan fungsi-fungsi utama, mulai dari ekstraksi data PDF menggunakan Regex, penyimpanan profil pelamar ke database MySQL, hingga pencarian data menggunakan algoritma KMP, Boyer-Moore, dan Levenshtein Distance.

Secara fungsional, aplikasi ini dapat digunakan sebagai alat bantu yang efektif untuk mempercepat proses penyaringan awal kandidat dalam proses rekrutmen.

b. Saran

Untuk pengembangan di masa depan, beberapa penyempurnaan dapat dipertimbangkan:

- Dukungan Format File: Menambah kemampuan untuk memproses format lain seperti `.docx`.
- Optimasi Pencarian: Mengimplementasikan algoritma Aho-Corasick untuk pencarian multi-kata kunci yang lebih efisien.
- Peningkatan Ekstraksi Data: Mengadopsi teknik *Natural Language Processing* (NLP) untuk ekstraksi informasi yang lebih akurat dan tidak bergantung pada format CV yang kaku.
- Pengembangan Platform: Membangun versi aplikasi berbasis web untuk meningkatkan aksesibilitas bagi tim.
- Penambahan Fitur: Melengkapi aplikasi dengan fungsionalitas untuk melacak status lamaran (misalnya, "screening", "interview") agar menjadi *Applicant Tracking System* (ATS) yang lebih lengkap.

Lampiran

- Tabel penyelesaian tugas besar

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	✓	
2	Aplikasi menggunakan basis data berbasis SQL dan berjalan dengan lancar.	✓	
3	Aplikasi dapat mengekstrak informasi penting menggunakan Regular Expression (Regex).	✓	
4	Algoritma *Knuth-Morris-Pratt (KMP)* dan *Boyer-Moore (BM)* dapat menemukan kata kunci dengan benar.	✓	
5	Algoritma Levenshtein Distance dapat mengukur kemiripan kata kunci dengan benar.	✓	
6	Aplikasi dapat menampilkan *summary CV applicant*.	✓	
7	Aplikasi dapat menampilkan *CV applicant* secara keseluruhan.	✓	
8	Membuat laporan sesuai dengan spesifikasi.	✓	
9	Membuat bonus enkripsi data profil *applicant*.		✓
10	Membuat bonus algoritma Aho-Corasick.		✓

- Tautan repository
  https://github.com/jandhiesto/Tubes3_LinkedOut

Daftar Pustaka

1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf
2. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf