

LAPORAN TUGAS BESAR 3
IF2211 STRATEGI ALGORITMA



oleh:
U_Tiga

Kenneth Ricardo Chandra	13523022
M. Kinan Arkansyaddad	13523152
Filbert Engyo	13523163

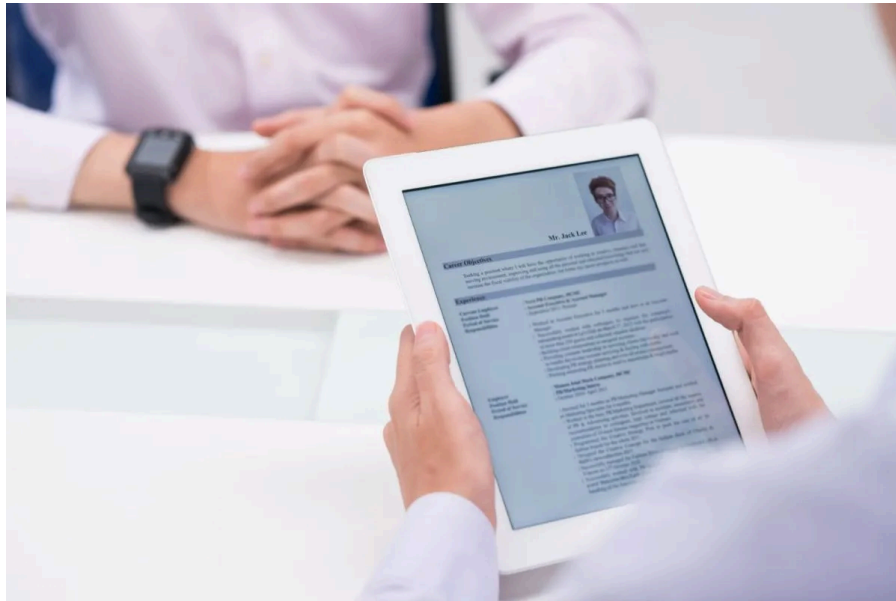
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB I	
DESKRIPSI TUGAS.....	3
BAB II	
LANDASAN TEORI.....	5
2.1. Algoritma Knuth-Morris-Pratt.....	5
2.2. Algoritma Boyer-Moore.....	5
2.3. Algoritma Aho-Corasick.....	5
2.4. Aplikasi Desktop.....	6
BAB III	
ANALISIS PEMECAHAN MASALAH.....	8
3.1. Langkah-langkah Pemecahan Persoalan.....	8
3.1.1. Parsing dari PDF.....	8
3.1.2. Menerima masukan dari pengguna.....	8
3.1.3. Mencari string yang berkaitan dengan masukan pengguna.....	8
3.1.4. Menampilkan hasil.....	9
3.2. Mapping Persoalan Menjadi Elemen-elemen Algoritma KMP dan BM.....	9
3.3. Fitur Fungsional dan Arsitektur Aplikasi Desktop yang Dibangun.....	9
3.4. Contoh Ilustrasi Kasus.....	9
BAB IV	
IMPLEMENTASI DAN PENGUJIAN.....	10
4.1. Spesifikasi Teknis Program.....	10
4.2. Penjelasan Tata Cara Penggunaan Program.....	39
4.3. Hasil Pengujian.....	40
4.4. Analisis Hasil.....	44
BAB V	
KESIMPULAN, SARAN, DAN REFLEKSI.....	46
A. Kesimpulan.....	46
B. Saran.....	46
C. Refleksi.....	46
LAMPIRAN.....	47
DAFTAR PUSTAKA.....	49

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.

BAB II

LANDASAN TEORI

2.1. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) adalah metode pencocokan string yang meningkatkan efisiensi algoritma brute-force dengan cara menghindari perbandingan karakter yang berulang. Prinsip utamanya adalah "pergeseran cerdas" (intelligent shift) setelah terjadi ketidakcocokan. Untuk melakukan ini, KMP terlebih dahulu memproses pola (pattern) untuk membuat sebuah tabel bantu yang disebut Fungsi Pinggiran (Border Function). Fungsi ini menyimpan panjang dari awalan (prefix) terpanjang dari sebuah sub-pola yang juga merupakan akhirannya (suffix), yang akan digunakan untuk menentukan sejauh mana pola harus digeser.

Selama proses pencocokan, yang berjalan dari kiri ke kanan, jika terjadi ketidakcocokan, KMP menggunakan tabel Fungsi Pinggiran untuk menggeser pola ke kanan. Keunikannya adalah penunjuk pada teks tidak pernah bergerak mundur. Sebaliknya, pola digeser maju untuk menyelaraskan prefix yang diketahui cocok dengan segmen teks yang baru saja dicocokkan, lalu perbandingan dilanjutkan dari titik ketidakcocokan. Pendekatan ini menghasilkan kompleksitas waktu yang linear, $O(m+n)$, dan menjadikannya sangat efisien untuk memproses data dari sumber seperti stream atau file yang sangat besar.

2.2. Algoritma Boyer-Moore

Algoritma Boyer-Moore (BM) adalah salah satu algoritma pencocokan string paling efisien dalam praktik, yang memperoleh kecepatannya dari pendekatan yang unik. Teknik utamanya adalah membandingkan pola dari kanan ke kiri (looking-glass technique). Ketika terjadi ketidakcocokan, BM memanfaatkan informasi dari karakter pada teks yang menyebabkan kegagalan tersebut, yang dikenal sebagai heuristik karakter buruk (bad character heuristic). Heuristik ini memungkinkan algoritma untuk melakukan pergeseran yang jauh lebih besar dari satu karakter.

Untuk mengimplementasikan pergeseran ini, Boyer-Moore membuat tabel Fungsi Kemunculan Terakhir (Last Occurrence Function) saat pra-pemrosesan. Tabel ini mencatat indeks kemunculan terakhir dari setiap karakter alfabet di dalam pola. Saat terjadi ketidakcocokan, pola digeser ke kanan untuk menyejajarkan "karakter buruk" pada teks dengan kemunculan terakhirnya di dalam pola. Jika karakter tersebut tidak ada dalam pola, seluruh pola dapat digeser melewatinya. Kemampuan untuk "melompati" sebagian besar teks inilah yang membuat Boyer-Moore sangat cepat, terutama pada teks dengan alfabet besar seperti bahasa alami.

2.3. Algoritma Aho-Corasick

Algoritma Aho-Corasick adalah sebuah metode pencocokan string multipola yang dirancang untuk menemukan semua kemunculan dari sekumpulan pola dalam sebuah teks secara

bersamaan. Konsep dasarnya secara cerdas menggabungkan struktur data Trie, yang menyimpan semua pola, dengan mekanisme fungsi kegagalan (failure function) yang diadaptasi dari algoritma KMP. Pada fase pra-pemrosesan, algoritma ini membangun Trie tersebut dan melengkapinya dengan "tautan kegagalan" (failure links) yang menghubungkan setiap node ke posisi alternatif yang mewakili prefiks terpanjang berikutnya yang mungkin cocok, serta menandai node-node yang merepresentasikan pola lengkap sebagai "output".

Selama fase pencocokan, teks input diproses hanya dalam satu kali lintasan, karakter per karakter, sambil melintasi Trie. Ketika terjadi ketidakcocokan, algoritma tidak memulai ulang dari awal, melainkan secara efisien mengikuti tautan kegagalan untuk beralih ke keadaan yang relevan, sehingga menghindari perbandingan yang tidak perlu. Keunggulan utama dari Aho-Corasick adalah kompleksitas waktunya yang tetap linear terlepas dari jumlah pola yang dicari, menjadikannya sangat unggul untuk aplikasi skala besar seperti deteksi intrusi jaringan, pemfilteran konten, dan analisis genomik.

2.4. Aplikasi Desktop

Library PyMySQL merupakan salah satu library Python yang berfungsi untuk menghubungkan aplikasi dengan database MySQL. Library ini bersifat pure Python, artinya tidak memerlukan komponen eksternal atau native C bindings, sehingga cukup ringan dan mudah digunakan untuk melakukan operasi dasar seperti eksekusi query SQL (SELECT, INSERT, UPDATE, DELETE). Fungsinya sangat penting dalam aplikasi desktop yang membutuhkan interaksi langsung dengan basis data, terutama jika ingin mengelola data pengguna, menyimpan hasil pemrosesan, atau membaca data yang relevan dengan operasi aplikasi.

Selanjutnya, PyQt6 adalah library antarmuka pengguna (GUI) yang menjadi tulang punggung tampilan aplikasi desktop ini. PyQt6 adalah binding Python dari framework Qt6, yang mendukung berbagai komponen antarmuka modern seperti jendela, tombol, form, dialog, hingga sistem event-driven. Dengan menggunakan PyQt6, developer dapat membangun aplikasi yang interaktif dan ramah pengguna tanpa harus menggunakan bahasa pemrograman C++ seperti pada Qt aslinya. Penggunaan PyQt6 sangat krusial dalam menciptakan pengalaman pengguna yang intuitif dan profesional pada aplikasi desktop.

Library lainnya yaitu mysql-connector-python, merupakan library resmi dari Oracle untuk koneksi ke MySQL. Fungsinya mirip dengan PyMySQL, namun memiliki keunggulan dalam dukungan fitur-fitur lanjutan dari MySQL serta kestabilan karena dikembangkan langsung oleh vendor resmi. Penggunaan dua library database (PyMySQL dan mysql-connector-python) ini bisa jadi pilihan fleksibel bagi developer untuk menyesuaikan dengan kebutuhan spesifik, baik untuk performa, kompatibilitas, atau preferensi struktur query.

Terakhir, PyMuPDF atau fitz, digunakan dalam aplikasi ini untuk menangani pemrosesan file PDF. Library ini sangat andal untuk membaca, mengekstrak teks, dan memanipulasi dokumen PDF, sehingga sangat cocok digunakan untuk aplikasi yang berkaitan dengan analisis dokumen seperti CV, laporan, atau formulir. Dalam konteks proyek ini, PyMuPDF kemungkinan besar digunakan untuk mengekstrak isi dari CV dalam format PDF, yang nantinya dianalisis atau disimpan ke database.

BAB III

ANALISIS PEMECAHAN MASALAH

Pada bab ini akan dibahas langkah-langkah dan komponen utama dalam merancang sistem CV ATS. Setiap poin menguraikan aspek penting mulai dari pemetaan masalah hingga penentuan strategi pencarian

3.1. Langkah-langkah Pemecahan Persoalan.

Untuk mempermudah proses pengembangan, maka persoalan ini dipecah secara berurutan sesuai seperti:

3.1.1. *Parsing* dari PDF

File CV dengan Format ATS yang dibuat dalam PDF akan dibaca dan dikonversi menjadi sebuah string panjang yang berisi seluruh isi teks dari dokumen tersebut, ini akan memanfaatkan *library* bawaan *fitz*.

3.1.2. Menerima masukan dari pengguna

Pengguna bisa memasukkan satu atau lebih *keyword* yang ingin dicari yang dipisah dengan koma apabila masukan lebih dari satu, lalu pengguna bisa memilih algoritma pencocokan yang diinginkan yaitu KMP / BM / Aho-corasick.

3.1.3. Mencari string yang berkaitan dengan masukan pengguna

Pemilihan algoritma pencocokan yang dipilih pengguna akan menentukan pemrosesan string *keyword* yang dimasukkan, *keyword* itu akan diproses terhadap hasil ekstrak dari PDF yang dilakukan sebelumnya untuk seluruh CV yang dibaca, proses ini dinamakan *exact matching* atau pencarian tepat. Apabila pencarian *exact matching* tidak ditemukan, maka program otomatis akan memanfaatkan metode *Levenshtein Distance* untuk melakukan pencarian khusus yang paling mirip dengan ambang batas kemiripan tertentu atau *fuzzy matching*.

3.1.4. Menampilkan hasil

CV yang paling mirip atau cocok dengan *keyword* setelah diproses dengan algoritma yang dipilih akan ditampilkan *summary* informasi secara berurutan berdasarkan frekuensi kemiripan *keyword*.

3.2. Mapping Persoalan Menjadi Elemen-elemen Algoritma KMP dan BM

Pemetaan masalah sebagai elemen algoritma KMP dan BM, serta Aho-corasick itu telah dijelaskan sebelumnya yaitu mencari sesuatu pada CV berdasarkan *keyword* tertentu. Proses berawal dari *parsing* isi PDF dari CV menjadi sebuah string panjang.. Algoritma KMP dan BM akan menjadi algoritma khususnya yang mencocokkan pola / *pattern* dalam informasi string tersebut, iterasi akan menghasilkan jumlah *match* pada suatu CV yang akan dibandingkan nantinya dengan CV lain dengan jumlah *match* yang lebih banyak.

3.3. Fitur Fungsional dan Arsitektur Aplikasi Desktop yang Dibangun.

Program dibuat berbasis pada arsitektur *monolithic* yang membagi kode-kode berdasarkan fungsionalitas yang berkaitan menjadi satu folder yang sama, dengan begitu program bisa lebih cepat dan mudah dibaca. Secara garis besar, program terbagi menjadi 6 bagian yaitu *algorithm* yang berisi algoritma yang digunakan untuk proses, *database* untuk koneksi dan inisialisasi basis data yang digunakan, *extractor* untuk proses *parsing* dan *extract* file dari pdf maupun hasilnya, *gui* yang berisi tampilan laman-laman program, *logic* yang berisi logika untuk *threading*, dan *utils* yang berisi mekanisme tambahan diluar algoritma dan *extractor*. Secara singkat, fitur-fitur yang disediakan dalam program ini meliputi:

- Membaca file CV dalam PDF dalam bentuk teks secara otomatis.
- Menyimpan data CV berupa nama, tanggal, lokasi file kedalam *database*
- Mencari CV berdasarkan yang *keyword* menggunakan algoritma Knuth-Morris-Pratt, Boyer-Moore, dan Aho-Corasick.
- Mencari CV berdasarkan alternatif dengan *fuzzy search* dengan algoritma Levenshtein Distance.
- Menampilkan *summary* CV yang menampilkan daftar education, skill, dan experience dengan regular expression.

3.4. Contoh Ilustrasi Kasus

Ilustrasi kasus yang terjadi adalah suatu perusahaan yang sedang melakukan pencocokan CV pelamar pekerjaan dengan kata kunci pada deskripsi pekerjaan. Maka dari itu, perusahaan menggunakan aplikasi ini untuk mencari kata kunci yang terdapat pada CV. Misalkan kata kuncinya “python”, “machine learning”, “docker”, dan “rest api”

Contoh Isi CV yang telah disimpan sebagai teks

“Experienced in Python and data analysis. Built several ML models. Familiar with Dockers, Flask framework, and RESTful APIs.”

Penggunaan Boyer-Moore digunakan untuk mencari satu pattern dalam satu teks dan untuk kasus ini, akan gagal saat mencari kata kunci Python karena terdapat typo Python. Boyer-Moore bekerja dari kanan ke kiri dan mengabaikan karakter awal jika tidak cocok. Penggunaan KMP dapat dilakukan karena untuk mencari rest api, terdapat substring yang mirip. Namun rest api dan RESTful APIs tidak cocok secara exact sehingga string matching akan gagal.

Untuk Aho-Corasick, akan mendeteksi Docker karena ada kemunculan Dockers, namun untuk kata-kata lain akan diabaikan karena terdapat perbedaan seperti typo, singkatan dan format yang berbeda. Maka, sebagian besar akan gagal untuk ditemukan tanpa fuzzy matching.

Penggunaan Levenshtein Distance akan bekerja misal pada kata python dan pyhton yang akan bernilai 0.667 karena dihitung menggunakan $1 - \text{LD}/\text{max lengthnya}$ yaitu $1 - 2/6 = 0.667$.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Spesifikasi Teknis Program

Program dibuat dan dibagi dalam beberapa bagian yaitu algoritma untuk melakukan string matching, database untuk menyimpan data, extractor untuk melakukan pengambilan data dari CV yang bersangkutan, GUI untuk tampilan aplikasi serta utils atau utility sebagai fungsi-fungsi pembantu dalam keseluruhan pembuatan aplikasi.

Algoritma yang digunakan untuk melakukan string-matching ada Boyer-Moore (BM), Knuth-Morris-Pratt(KMP) dan juga Aho-Corasick serta Levenshtein Distance untuk mengukur tingkat kemiripan dari kata-kata yang dicari. Berikut merupakan implementasinya.

[bm.py](#)

```
class BM:
    def __init__(self, pattern: str):
        self.pattern = pattern
        self.m = len(pattern)
        if self.m > 0:
            self.bad_char_table = self._build_bad_char_table()
            self.good_suffix_shifts = self._build_good_suffix_table()

    def _build_bad_char_table(self) -> dict:
```

```

bad_char = {}
for i in range(self.m):
    bad_char[self.pattern[i]] = i
return bad_char

def _build_good_suffix_table(self) -> list[int]:
    m = self.m
    shifts = [0] * (m + 1)
    border_pos = [0] * (m + 1)

    i = m
    j = m + 1
    border_pos[i] = j
    while i > 0:
        while j <= m and self.pattern[i - 1] != self.pattern[j - 1]:
            if shifts[j] == 0:
                shifts[j] = j - i
            j = border_pos[j]
        i -= 1
        j -= 1
        border_pos[i] = j

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

    return shifts

def search(self, text: str) -> list[int]:
    n = len(text)
    m = self.m

    if m == 0 or n == 0 or m > n:
        return []

    occurrences = []
    s = 0

    while s <= (n - m):
        j = m - 1

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

        if j < 0:
            occurrences.append(s)
            s += self.good_suffix_shifts[0]
        else:
            char_in_text = text[s + j]
            last_occurrence = self.bad_char_table.get(char_in_text, -1)

            bad_char_shift = j - last_occurrence

```

```

        good_suffix_shift = self.good_suffix_shifts[j + 1]

        s += max(1, bad_char_shift, good_suffix_shift)

    return occurrences

def count_occurrences(self, text: str) -> int:
    return len(self.search(text))

```

[kmp.py](#)

```

class KMP:
    def __init__(self, pattern: str):
        self.pattern = pattern
        self.prefix_table = self._build_prefix_table()

    def _build_prefix_table(self) -> list[int]:
        m = len(self.pattern)
        lps = [0] * m
        length = 0
        i = 1

        while i < m:
            if self.pattern[i] == self.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 search(self, text: str) -> list[int]:
        n = len(text)
        m = len(self.pattern)

        if m == 0: return []
        if n == 0: return []
        if m > n: return []

        lps = self.prefix_table
        occurrences = []

        i = 0
        j = 0

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

```

```

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

    return occurrences

def count_occurrences(self, text: str) -> int:
    return len(self.search(text))

```

aho_corasick.py

```

import collections

class TrieNode:
    """
    Represents a single node in the Aho-Corasick trie.
    Each node is a state in the finite automaton.
    """
    def __init__(self, node_id):
        self.id = node_id

        self.children = {}

        self.output = []

        self.failure = 0

        self.parent = None

class AhoCorasick:
    """
    An implementation of the Aho-Corasick algorithm for efficient multi-pattern string matching.
    It builds a finite automaton from a set of keywords and then processes a text in a single pass.
    """
    def __init__(self):
        self.nodes = [TrieNode(0)]
        self._node_count = 1
        self._finalized = False

    def _get_new_node(self, parent_id=None):
        """Helper to create a new node and add it to our list of nodes."""
        new_node = TrieNode(self._node_count)
        new_node.parent = parent_id
        self.nodes.append(new_node)
        self._node_count += 1
        return new_node.id

    def add_pattern(self, pattern: str):

```

```

"""
Inserts a pattern into the trie.
This builds the basic keyword tree structure.
"""
if not pattern:
    return

if self._finalized:
    raise ValueError("Cannot add new patterns after building failure links.")

current_node_id = 0
for char in pattern:
    current_node = self.nodes[current_node_id]
    if char not in current_node.children:
        new_node_id = self._get_new_node(parent_id=current_node_id)
        current_node.children[char] = new_node_id
    current_node_id = current_node.children[char]

# The pattern ends at this node, so add it to the output list.
self.nodes[current_node_id].output.append(pattern)

def build_failure_links(self):
    """
    Constructs the failure links for the entire trie.
    This must be called after all patterns have been added and before searching.
    This process uses a Breadth-First Search (BFS) approach.
    """
    queue = collections.deque()

    root = self.nodes[0]
    for child_id in root.children.values():
        self.nodes[child_id].failure = 0
        queue.append(child_id)

    while queue:
        current_node_id = queue.popleft()
        current_node = self.nodes[current_node_id]

        for char, next_node_id in current_node.children.items():
            failure_id = current_node.failure

            while char not in self.nodes[failure_id].children and failure_id != 0:
                failure_id = self.nodes[failure_id].failure

            if char in self.nodes[failure_id].children:
                self.nodes[next_node_id].failure = self.nodes[failure_id].children[char]
            else:
                self.nodes[next_node_id].failure = 0

            failure_output_node_id = self.nodes[next_node_id].failure
            self.nodes[next_node_id].output.extend(self.nodes[failure_output_node_id].output)

            queue.append(next_node_id)

    self._finalized = True

```

```

def search(self, text: str) -> list[tuple[int, str]]:
    """
    Searches for all added patterns within the given text.

    Args:
        text: The string to search within.

    Returns:
        A list of tuples, where each tuple contains (start_index, matched_pattern).
        The list is not guaranteed to be in any specific order.
    """
    if not self._finalized:
        raise ValueError("Failure links must be built before searching. Call build_failure_links().")

    results = []
    current_node_id = 0

    for i, char in enumerate(text):
        while char not in self.nodes[current_node_id].children and current_node_id != 0:
            current_node_id = self.nodes[current_node_id].failure

        if char in self.nodes[current_node_id].children:
            current_node_id = self.nodes[current_node_id].children[char]
        else:
            current_node_id = 0

        if self.nodes[current_node_id].output:
            for pattern in self.nodes[current_node_id].output:
                start_index = i - len(pattern) + 1
                results.append((start_index, pattern))

    return results

```

levenshtein.py

```

class Levenshtein:
    def __init__(self, threshold: float = 0.8):
        self.threshold = threshold

    def _calculate_distance(self, s1: str, s2: str) -> int:
        len_s1 = len(s1)
        len_s2 = len(s2)

        dp = [[0] * (len_s2 + 1) for _ in range(len_s1 + 1)]

        for i in range(len_s1 + 1):
            dp[i][0] = i
        for j in range(len_s2 + 1):
            dp[0][j] = j

        for i in range(1, len_s1 + 1):
            for j in range(1, len_s2 + 1):

```

```

        cost = 0 if s1[i-1] == s2[j-1] else 1

        dp[i][j] = min(
            dp[i-1][j] + 1,
            dp[i][j-1] + 1,
            dp[i-1][j-1] + cost
        )

    return dp[len_s1][len_s2]

def similarity(self, word1: str, word2: str) -> float:
    distance = self._calculate_distance(word1.lower(), word2.lower())
    max_len = max(len(word1), len(word2))

    if max_len == 0:
        return 1.0

    return 1.0 - (distance / max_len)

def is_similar(self, word1: str, word2: str) -> bool:
    return self.similarity(word1, word2) >= self.threshold

def find_similar_keywords(self, keywords: list[str], text: str) -> dict[str, list[int]]:
    words_in_text = text.lower().split()

    similar_matches = {}
    for keyword in keywords:
        keyword_lower = keyword.lower()
        matching_indices = []

        for i, text_word in enumerate(words_in_text):
            if self.is_similar(keyword_lower, text_word):
                matching_indices.append(i)

        if matching_indices:
            similar_matches[keyword] = matching_indices

    return similar_matches

```

Setelah bagian algoritma, terdapat bagian database yang akan membuat tabel dalam mysql dengan tujuan untuk menyimpan informasi-informasi penting yang nantinya akan ditampilkan melalui GUI. Berikut merupakan implementasinya

db_init.py

```

import pymysql

def get_connection():
    """Mengembalikan koneksi ke database MySQL menggunakan PyMySQL."""
    return pymysql.connect(
        host="localhost",
        user="root",

```



```

        password="",
        database="ats_db",
        cursorclass=pymysql.cursors.Cursor,
        autocommit=False
    )

def init_db():
    """Membuat tabel ApplicantProfile dan ApplicationDetail jika belum ada."""
    conn = get_connection()
    cursor = conn.cursor()

    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
);
""")

    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)
);
""")

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

```

[models.py](#)

```

from dataclasses import dataclass
from typing import Optional

@dataclass
class ApplicantProfile:
    applicant_id: Optional[int] = None
    first_name: str = ""
    last_name: str = ""
    date_of_birth: Optional[str] = None
    address: str = ""
    phone_number: str = ""

@dataclass
class ApplicationDetail:
    detail_id: Optional[int] = None

```

```
applicant_id: int = None
application_role: str = ""
cv_path: str = ""
```

[query.py](#)

```
from .db_init import get_connection
from .models import ApplicantProfile, ApplicationDetail

def insert_applicant(profile: ApplicantProfile) -> int:
    """Menyimpan profil applicant, return id yang di-generate."""
    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        INSERT INTO ApplicantProfile (first_name, last_name, date_of_birth, address, phone_number)
        VALUES (%s, %s, %s, %s, %s)
    """, (profile.first_name, profile.last_name, profile.date_of_birth, profile.address,
    profile.phone_number))

    applicant_id = cursor.lastrowid
    conn.commit()
    cursor.close()
    conn.close()
    return applicant_id

def insert_application(detail: ApplicationDetail) -> int:
    """Menyimpan detail lamaran dan path CV."""
    conn = get_connection()
    cursor = conn.cursor()

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

    detail_id = cursor.lastrowid
    conn.commit()
    cursor.close()
    conn.close()
    return detail_id

def get_all_applications() -> list[tuple]:
    """Mengambil semua aplikasi dan CV path-nya."""
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("""
        SELECT ap.applicant_id, ap.first_name, ap.last_name, ap.date_of_birth, ap.address,
        ap.phone_number, ad.cv_path, ad.application_role
        FROM ApplicantProfile ap
        JOIN ApplicationDetail ad ON ap.applicant_id = ad.applicant_id
    """)
    results = cursor.fetchall()
```

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

Setelah bagian database, terdapat bagian extractor yang berfungsi untuk melakukan ekstraksi data pada pdf atau cv yang akan dianalisa.

cv_processor.py

```
from src.extractor.pdf_parser import PDFParser
from src.extractor.regex_extractor import RegexExtractor
from typing import Any

class CVProcessor:
    def __init__(self, pdf_path: str):
        self.pdf_path = pdf_path
        self.text = ""

    def process(self) -> str:
        """Mengekstrak teks dari PDF dan mengambil informasi penting."""
        self.text = PDFParser(self.pdf_path).extract_text()
        return RegexExtractor(self.text).extract_all()

    def get_raw_text(self) -> str:
        """Return teks CV mentah setelah diekstrak dari PDF."""
        if not self.text:
            self.text = PDFParser(self.pdf_path).extract_text()
        return self.text
```

pdf_parser.py

```
import fitz

class PDFParser:
    def __init__(self, pdf_path: str):
        self.pdf_path = pdf_path

    def extract_text(self) -> str:
        """
        Membaca seluruh halaman PDF dan menggabungkannya menjadi satu string.
        Return: teks lengkap dari dokumen.
        """
        text = ""
        with fitz.open(self.pdf_path) as doc:
            for page in doc:
                text += page.get_text()
        return text
```

regex_extractor.py

```
import re
from typing import Dict, List, Any

class RegexExtractor:
    def __init__(self, raw_text: str):
        self.text = raw_text

    def extract_all(self) -> str:
        text = self.text
        all_info = {}

        text = re.sub(r'\s{2,}', ' ', text)
        text = re.sub(r'\n{2,}', '\n', text)

        job_title_pattern = r'^(?:--- PAGE \d+
---\n)?\s*([A-Z][A-Z\s.-]+)(?=\n[A-Z][a-z]+\nSkill\nSummary\nObjective|$)'
        job_title_match = re.search(job_title_pattern, text, re.MULTILINE)
        if job_title_match:
            all_info['Job Title'] = job_title_match.group(1).strip()
            all_info['Job Title'] = re.sub(r'[•Â$]', ' ', all_info['Job Title']).strip()
            all_info['Job Title'] = re.sub(r'\s+', ' ', all_info['Job Title']).strip()

        #Cari skills
        skills_pattern = r'Skills\s*\n*(.*?)(?=\n\s*[A-Z][a-z]+\s*\n|Z)'
        skills_match = re.search(skills_pattern, text, re.DOTALL | re.IGNORECASE)
        if skills_match:
            skills_text = skills_match.group(1).strip()
            skills_text = re.sub(r'\n+', ' ', skills_text)
            skills_text = re.sub(r'\s+', ' ', skills_text)
            skills_text = re.sub(r'[-.]', ' ', skills_text).strip()
            all_info['Skills'] = skills_text

        #Cari summary
        summary_pattern = r'Summary\s*\n*(.*?)(?=\n\s*[A-Z][a-z]+\s*\n|$)'
        summary_match = re.search(summary_pattern, text, re.DOTALL | re.IGNORECASE)
        if summary_match:
            summary_text = summary_match.group(1).strip()
            summary_text = re.sub(r'\n+', ' ', summary_text)
            summary_text = re.sub(r'\s+', ' ', summary_text)
            summary_text = re.sub(r'[•Â$]', ' ', summary_text).strip()
            all_info['Summary'] = summary_text

        #Cari highlights
        highlights_pattern = r'Highlights\s*\n*(.*?)(?=\n\s*[A-Z][a-z]+\s*\n|$)'
        highlights_match = re.search(highlights_pattern, text, re.DOTALL | re.IGNORECASE)
        if highlights_match:
            highlights_text = highlights_match.group(1).strip()
            highlights_text = re.sub(r'\n+', ' ', highlights_text)
            highlights_text = re.sub(r'\s+', ' ', highlights_text)
            highlights_text = re.sub(r'[•Â$]', ' ', highlights_text).strip()
            all_info['Highlights'] = highlights_text

        #Cari accomplishments
```

```

accomplishments_pattern = r'Accomplishments\s*\n+(.*?)(?=\n\s*[A-Z][a-z]+\s*\n|$)'
accomplishments_match = re.search(accomplishments_pattern, text, re.DOTALL |
re.IGNORECASE)
if accomplishments_match:
    accomplishments_text = accomplishments_match.group(1).strip()
    accomplishments_text = re.sub(r'\n+', ' ', accomplishments_text)
    accomplishments_text = re.sub(r'\s+', ' ', accomplishments_text)
    accomplishments_text = re.sub(r'[\u201c\u201d]', '', accomplishments_text).strip()
    all_info['Accomplishments'] = accomplishments_text

#Cari work experiences
experience_pattern =
r'Experience\s*\n+(.*?)(?=\n\s*Education|\n\s*Certifications|\n\s*Interests|\n\s*Additional
Information|\Z)'
experience_match = re.search(experience_pattern, text, re.DOTALL | re.IGNORECASE)

final_job_entries = []

if experience_match:
    experience_block = experience_match.group(1).strip()
    experience_block = re.sub(r'\n{2,}', '\n', experience_block)

    job_entry_pattern =
r'(\d{2}/\d{4})\s*(?:to|-)\s*(?:Current|Present|\d{2}/\d{4}))\s*(.*?)(?=\n*\d{2}/\d{4})\s*(?:to|-)\s*(?:Curr
ent|Present|\d{2}/\d{4})\Z)'
    raw_job_matches = re.findall(job_entry_pattern, experience_block, re.DOTALL |
re.IGNORECASE)

    for date_range, job_content_block in raw_job_matches:
        current_job_dict = {
            'Date': date_range.strip(),
            'Company': 'N/A',
            'Position': 'N/A',
        }

        lines_in_block = [line.strip() for line in job_content_block.split('\n') if line.strip()]

        if lines_in_block:
            company_found_index = -1
            for i, line_content in enumerate(lines_in_block):
                if "Company Name" in line_content or re.match(r'^[A-Z][a-zA-Z\s-]+(?: LLC| Inc\.|
Corp\.)?$', line_content):
                    current_job_dict['Company'] = re.sub(r'(? - City, State| City, State)?', '',
line_content).strip()
                    current_job_dict['Company'] = re.sub(r'[\u201c\u201d]', '', current_job_dict['Company']).strip()
                    company_found_index = i
                    break

            start_desc_index = company_found_index + 1 if company_found_index != -1 else 0

            full_pos_desc_text = " ".join(lines_in_block[start_desc_index:]).strip()
            full_pos_desc_text = re.sub(r'\n+', ' ', full_pos_desc_text)
            full_pos_desc_text = re.sub(r'\s+', ' ', full_pos_desc_text)
            full_pos_desc_text = re.sub(r'[\u201c\u201d]', '', full_pos_desc_text).strip()

```

```

        current_job_dict['Position'] = full_pos_desc_text if full_pos_desc_text else 'N/A'

        if current_job_dict['Company'] == 'N/A' and lines_in_block:
            first_significant_line = lines_in_block[0]
            if len(first_significant_line.split()) < 7 and re.match(r'^[A-Z][a-zA-Z\s.-]+$',
first_significant_line):
                current_job_dict['Company'] = re.sub(r'(? : - City, State| City, State)?', '',
first_significant_line).strip()
                current_job_dict['Company'] = re.sub(r'[•Â$]', '', current_job_dict['Company']).strip()
                current_job_dict['Position'] = " ".join(lines_in_block[1:]).strip()
                current_job_dict['Position'] = re.sub(r'\n+', ' ', current_job_dict['Position'])
                current_job_dict['Position'] = re.sub(r'\s+', ' ', current_job_dict['Position'])
                current_job_dict['Position'] = re.sub(r'[•Â$]', '', current_job_dict['Position']).strip()
            else:
                current_job_dict['Position'] = first_significant_line
                current_job_dict['Position'] += " ".join(lines_in_block[1:]).strip()
                current_job_dict['Position'] = re.sub(r'\n+', ' ', current_job_dict['Position'])
                current_job_dict['Position'] = re.sub(r'\s+', ' ', current_job_dict['Position'])
                current_job_dict['Position'] = re.sub(r'[•Â$]', '', current_job_dict['Position']).strip()

        final_job_entries.append(current_job_dict)

#Cari education
education_pattern =
r'Education(?:\s+and\s+Training)?\s*\n+(.*)?(?=\n\s*Skills\n\s*Certifications\n\s*Interests\n\s*Additio
nal Information\Z)'
education_match = re.search(education_pattern, text, re.DOTALL | re.IGNORECASE)
if education_match:
    education_text = education_match.group(1).strip()

    school_pattern =
r'([A-Za-z\s.-]+(?:College|University|School|Institution))\s*(?:$[^\n]*(?:City,\s*State))?'
    school_match = re.search(school_pattern, education_text, re.IGNORECASE)

    degree_pattern = r'(?:(Associate|Bachelor|Master|PhD|RN Diploma|High School Diploma|Post
Secondary Training
Certificate|Diploma).*\s*)([A-Za-z\s]+(?:\s*in\s*[A-Za-z\s]+)?(?:/\s*[A-Za-z\s]+)?(?:-\s*[A-Za-z\s]+
)?)'
    degree_match = re.search(degree_pattern, education_text, re.IGNORECASE)

    year_pattern = r'\b(19|20)\d{2}\b'
    years = re.findall(year_pattern, education_text)

    education_clean = education_text
    education_clean = re.sub(r'\n+', ' ', education_clean)
    education_clean = re.sub(r'\s+', ' ', education_clean)
    education_clean = re.sub(r'[•Â$]', '', education_clean).strip()

    degree_str = None
    if degree_match:
        if degree_match.group(1):
            degree_str = f'{degree_match.group(1).strip()} in {degree_match.group(2).strip()}'
        else:
            degree_str = degree_match.group(2).strip()
    degree_str = re.sub(r'^[•\W]+', '', degree_str).strip()

```

```

        degree_str = re.sub(r'[\W]+\$', " ", degree_str).strip()

    all_info['Education'] = {
        'Full_Text': education_clean,
        'School': school_match.group(1).strip() if school_match else None,
        'Degree': degree_str,
        'Years': ', '.join(sorted(list(set(years)))) if years else None
    }

    #Cari certifications / licenses
    certifications_pattern =
r'Certifications(?:\s*\V\s*and\s+)?(?:Licenses)?\s*\n+(.*?)(?=\n\s*[A-Z][a-z]+\s*\n\Z)'
    certifications_match = re.search(certifications_pattern, text, re.DOTALL | re.IGNORECASE)
    if certifications_match:
        certifications_text = certifications_match.group(1).strip()
        certifications_text = re.sub(r'\n+', ' ', certifications_text)
        certifications_text = re.sub(r'\s+', ' ', certifications_text)
        certifications_text = re.sub(r'[\u00c0\u00a0]', " ", certifications_text).strip()
        all_info['Certifications'] = certifications_text

    all_skill_parts = []

    if 'Skills' in all_info and all_info['Skills']:
        if not re.match(r'Skills Used|I enjoy|my best|working as a team|if needed', all_info['Skills'],
re.IGNORECASE):
            all_skill_parts.append(all_info['Skills'])

    for job in final_job_entries:
        if job.get('Position') and job['Position'] != 'N/A':
            resp_keyword_match = re.search(r'(?Responsibilities|Duties)\s*(.*)', job['Position'],
re.DOTALL | re.IGNORECASE)
            if resp_keyword_match:
                all_skill_parts.append(resp_keyword_match.group(1).strip())
            else:
                all_skill_parts.append(job['Position'])

    if 'Certifications' in all_info and all_info['Certifications']:
        all_skill_parts.append(all_info['Certifications'])

    combined_skills_string = " ".join(all_skill_parts)
    combined_skills_string = re.sub(r'\s{2,}', ' ', combined_skills_string)
    combined_skills_string = re.sub(r'[\u00c0\u00a0]', ' ', combined_skills_string)
    combined_skills_string = re.sub(r'\s{2,}', ' ', combined_skills_string).strip()

    all_info['Consolidated_Skills'] = combined_skills_string

    # Format summary untuk CV
    result = "" * 30 + "\n"
    result += f"                                CV Summary\n"
    result += "" * 30 + "\n\n"

    if 'Job_Title' in all_info:
        result += "JOB TITLE:\n"
        result += f" {all_info['Job_Title']}\n\n"

```

```

if 'Consolidated_Skills' in all_info and all_info['Consolidated_Skills']:
    result += "SKILLS:\n"
    result += f" {all_info['Consolidated_Skills']}\n\n"

if 'Summary' in all_info:
    result += "SUMMARY:\n"
    result += f" {all_info['Summary']}\n\n"

if 'Highlights' in all_info:
    result += "HIGHLIGHTS:\n"
    result += f" {all_info['Highlights']}\n\n"

if 'Accomplishments' in all_info:
    result += "ACCOMPLISHMENTS:\n"
    result += f" {all_info['Accomplishments']}\n\n"

if final_job_entries:
    result += "WORK EXPERIENCE:\n"
    for i, entry in enumerate(final_job_entries):
        result += f" Date : {entry.get('Date', 'N/A')}\n"
        result += f" Company : {entry.get('Company', 'N/A')}\n"
        result += f" Position : {entry.get('Position', 'N/A')}\n"
        if i < len(final_job_entries) - 1:
            result += f" {'.' * 50}\n"
    result += "\n"

if 'Education' in all_info:
    result += "EDUCATION:\n"
    edu = all_info['Education']

    if edu['School']:
        result += f" School : {edu['School']}\n"
    if edu['Degree']:
        result += f" Degree : {edu['Degree']}\n"
    if edu['Years']:
        result += f" Years : {edu['Years']}\n"

    result += f" Full Info: {edu['Full_Text']}\n\n"

print(result)
return result

```

Bagian GUI terdiri dari beberapa bagian, yaitu main_window sebagai halaman utama, result_card sebagai laman hasil, search_page untuk pencarian kata, dan summary_page sebagai hasil pencarian serta rangkuman dari pencarian yang dilakukan. GUI ini

main_window.py

```

from PyQt6.QtWidgets import QMainWindow, QStackedWidget
from .search_page import SearchPage
from .summary_page import SummaryPage

```



```

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("CV Analyzer App")
        self.setGeometry(100, 100, 900, 700)

        self.stacked_widget = QStackedWidget()
        self.setCentralWidget(self.stacked_widget)

        self.search_page = SearchPage(self)
        self.summary_page = SummaryPage(self)

        self.stacked_widget.addWidget(self.search_page)
        self.stacked_widget.addWidget(self.summary_page)

        self.show_search_page()

    def show_search_page(self):
        self.stacked_widget.setCurrentWidget(self.search_page)

    def show_summary_page(self, data=None):
        if data:
            self.summary_page.load_data(data)
            self.stacked_widget.setCurrentWidget(self.summary_page)

```

result_card.py

```

from PyQt6.QtWidgets import QFrame, QVBoxLayout, QHBoxLayout, QLabel, QPushButton, QWidget
from PyQt6.QtGui import QFont
from PyQt6.QtCore import pyqtSignal, Qt

class ResultCard(QFrame):
    """
    A custom widget that displays a single applicant's search result in a card format.
    It emits signals when its action buttons are clicked.
    """
    # Custom signal
    summary_requested = pyqtSignal(dict)
    view_cv_requested = pyqtSignal(str)

    def __init__(self, applicant_data: dict, parent=None):
        super().__init__(parent)
        self.setObjectName("ResultCard")
        self.applicant_data = applicant_data
        self.init_ui()

    def init_ui(self):
        self setFrameShape(QFrame.Shape.StyledPanel) # Provides a border and background
        self setFrameShadow(QFrame.Shadow.Raised) # Adds a shadow effect
        self.setMinimumHeight(120)

        # Main Layout
        main_layout = QVBoxLayout(self)

```

```

# Top Section (Name and Match Count)
top_layout = QHBoxLayout()

name_label = QLabel(self.applicant_data.get('name', 'Unknown Applicant'))
name_label.setFont(QFont("Helvetica", 12, QFont.Weight.Bold))
top_layout.addWidget(name_label)

top_layout.addStretch()

match_count = self.applicant_data.get('match_count', 0)
match_label = QLabel(f'{match_count} Matches')
top_layout.addWidget(match_label)

main_layout.addLayout(top_layout)

# Middle Section (Matched Keywords)
keywords = self.applicant_data.get('matched_keywords', {})
keywords_text = "Found on: " + ", ".join(keywords.keys())
keywords_label = QLabel(keywords_text)
keywords_label.setWordWrap(True)
main_layout.addWidget(keywords_label)

main_layout.addStretch()

# Bottom Section (Action Buttons)
button_layout = QHBoxLayout()
button_layout.addStretch() # Pushes buttons to the right

summary_button = QPushButton("View Summary")
summary_button.setCursor(Qt.CursorShape.PointingHandCursor)
summary_button.clicked.connect(self.emit_summary_request)
button_layout.addWidget(summary_button)

view_cv_button = QPushButton("View CV File")
view_cv_button.setCursor(Qt.CursorShape.PointingHandCursor)
view_cv_button.clicked.connect(self.emit_view_cv_request)
button_layout.addWidget(view_cv_button)

main_layout.addLayout(button_layout)

def emit_summary_request(self):
    self.summary_requested.emit(self.applicant_data)

def emit_view_cv_request(self):
    self.view_cv_requested.emit(self.applicant_data.get('cv_path', ''))

```

search_page.py

```

import sys
import math
from PyQt6.QtWidgets import (QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit,
                             QPushButton, QRadioButton, QComboBox, QGroupBox,
                             QMessageBox, QScrollArea, QSpinBox)
from PyQt6.QtGui import QFont, QDesktopServices

```

```

from PyQt6.QtCore import QUrl, Qt, QThread
import os

from .result_card import ResultCard
from src.logic.search_worker import SearchWorker

class SearchPage(QWidget):
    def __init__(self, controller):
        super().__init__()
        self.controller = controller

        # State Variables
        self.all_results = []
        self.current_page = 1
        self.results_per_page = 10
        self.search_thread = None
        self.search_worker = None

        self.init_ui()

    def init_ui(self):
        main_layout = QVBoxLayout(self)

        # Title
        title_label = QLabel("CV Analyzer App")
        title_label.setFont(QFont("Helvetica", 18, QFont.Weight.Bold))
        main_layout.addWidget(title_label)

        # Search Criteria GroupBox
        input_group = QGroupBox()
        input_layout = QVBoxLayout(input_group)

        # Keywords Layout
        keywords_layout = QHBoxLayout()
        keywords_layout.addWidget(QLabel("Keywords (comma-separated):"))
        self.keywords_entry = QLineEdit()
        self.keywords_entry.setPlaceholderText("e.g., Python, React, SQL")
        keywords_layout.addWidget(self.keywords_entry)
        input_layout.addLayout(keywords_layout)

        options_layout = QVBoxLayout()

        # Algorithm Group
        options_layout.addWidget(QLabel("Algorithm:"))
        self.kmp_radio = QRadioButton("KMP")
        self.bm_radio = QRadioButton("BM")
        self.aho_radio = QRadioButton("Aho-Corasick")
        self.aho_radio.setChecked(True)
        options_layout.addWidget(self.kmp_radio)
        options_layout.addWidget(self.bm_radio)
        options_layout.addWidget(self.aho_radio)

        dropdown_layout = QHBoxLayout()

        # Top Matches Group

```

```

dropdown_layout.addWidget(QLabel("Top Matches:"))
self.top_n_spinbox = QSpinBox()
self.top_n_spinbox.setMinimum(1)
self.top_n_spinbox.setMaximum(1000)
self.top_n_spinbox.setValue(25)
dropdown_layout.addWidget(self.top_n_spinbox)

# Results Per Page Group
dropdown_layout.addWidget(QLabel("Results Per Page:"))
self.results_per_page_combo = QComboBox()
self.results_per_page_combo.addItem("5", "10", "20", "50"])
self.results_per_page_combo.setCurrentText("10")
dropdown_layout.addWidget(self.results_per_page_combo)

options_layout.addLayout(dropdown_layout)

input_layout.addLayout(options_layout)
main_layout.addWidget(input_group)

# Search and Cancel Buttons
search_button_layout = QHBoxLayout()
self.search_button = QPushButton("Search")
self.search_button.clicked.connect(self.on_search_clicked)
self.cancel_button = QPushButton("Cancel")
self.cancel_button.clicked.connect(self.on_cancel_clicked)
self.cancel_button.setEnabled(False)
search_button_layout.addWidget(self.search_button)
search_button_layout.addWidget(self.cancel_button)
main_layout.addLayout(search_button_layout)

# Results GroupBox
results_group = QGroupBox()
results_group_layout = QVBoxLayout(results_group)
main_layout.addWidget(results_group, 1)

self.summary_label = QLabel("Ready to search.")
results_group_layout.addWidget(self.summary_label)

scroll_area = QScrollArea()
scroll_area.setWidgetResizable(True)

self.card_container = QWidget()
self.card_layout = QVBoxLayout(self.card_container)
self.card_layout.setSpacing(10)
self.card_layout.addStretch()

scroll_area.setWidget(self.card_container)
results_group_layout.addWidget(scroll_area)

# Pagination Controls
pagination_layout = QHBoxLayout()
self.prev_button = QPushButton("< Previous")
self.prev_button.setEnabled(False)
self.prev_button.clicked.connect(self.go_to_prev_page)

```

```

self.page_label = QLabel("Page 1 / 1")
self.page_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

self.next_button = QPushButton("Next >")
self.next_button.setEnabled(False)
self.next_button.clicked.connect(self.go_to_next_page)

pagination_layout.addStretch()
pagination_layout.addWidget(self.prev_button)
pagination_layout.addWidget(self.page_label)
pagination_layout.addWidget(self.next_button)
pagination_layout.addStretch()

results_group_layout.addLayout(pagination_layout)

def on_search_clicked(self):
    keywords = [k.strip() for k in self.keywords_entry.text().split(',') if k.strip()]
    if not keywords:
        QMessageBox.warning(self, "Input Error", "Please enter at least one keyword.")
        return

    if self.kmp_radio.isChecked():
        algorithm = "KMP"
    elif self.bm_radio.isChecked():
        algorithm = "BM"
    else:
        algorithm = "AhoCorasick"

    top_n = self.top_n_spinbox.value()
    self.results_per_page = int(self.results_per_page_combo.currentText())

    self.search_button.setEnabled(False)
    self.cancel_button.setEnabled(True)
    self.summary_label.setText("Searching... Please wait.")

    self.search_thread = QThread()
    self.search_worker = SearchWorker(keywords, algorithm, top_n)
    self.search_worker.moveToThread(self.search_thread)

    self.search_thread.started.connect(self.search_worker.run)
    self.search_worker.search_finished.connect(self.on_search_finished)
    self.search_worker.search_error.connect(self.on_search_error)

    self.search_thread.start()

def on_cancel_clicked(self):
    if self.search_worker:
        self.search_worker.cancel()
    if self.search_thread:
        self.search_thread.quit()
        self.search_thread.wait()
    self.summary_label.setText("Search cancelled.")
    self.search_button.setEnabled(True)
    self.cancel_button.setEnabled(False)

```

```

def on_search_finished(self, results, exact_time, fuzzy_time):
    self.all_results = results
    time_text = f"Exact Match ({self.search_worker.algorithm}): {exact_time:.2f} ms"
    if fuzzy_time > 0:
        time_text += f" | Fuzzy Match: {fuzzy_time:.2f} ms"
    self.summary_label.setText(f"Found {len(self.all_results)} total results. {time_text}")

    self.search_button.setEnabled(True)
    self.cancel_button.setEnabled(False)
    self.current_page = 1
    self.update_page_display()

    self.search_thread.quit()
    self.search_thread.wait()

def on_search_error(self, error_message):
    QMessageBox.critical(self, "Search Error", f"An unexpected error occurred: {error_message}")
    self.all_results = []
    self.search_button.setEnabled(True)
    self.cancel_button.setEnabled(False)
    self.summary_label.setText("Search failed.")

    self.search_thread.quit()
    self.search_thread.wait()

    self.update_page_display()

def update_page_display(self):
    self.clear_layout(self.card_layout)

    total_results = len(self.all_results)
    if total_results == 0:
        self.page_label.setText("Page 0 / 0")
        self.prev_button.setEnabled(False)
        self.next_button.setEnabled(False)
        no_results_label = QLabel("No matching applicants found.")
        self.card_layout.insertWidget(0, no_results_label)
        return

    total_pages = math.ceil(total_results / self.results_per_page)
    start_index = (self.current_page - 1) * self.results_per_page
    end_index = start_index + self.results_per_page

    page_results = self.all_results[start_index:end_index]

    for res_data in page_results:
        card = ResultCard(res_data)
        card.summary_requested.connect(self.handle_summary_request)
        card.view_cv_requested.connect(self.handle_view_cv_request)
        self.card_layout.insertWidget(self.card_layout.count() - 1, card)

    self.page_label.setText(f"Page {self.current_page} / {total_pages}")
    self.prev_button.setEnabled(self.current_page > 1)
    self.next_button.setEnabled(self.current_page < total_pages)

```

```

def go_to_prev_page(self):
    if self.current_page > 1:
        self.current_page -= 1
        self.update_page_display()

def go_to_next_page(self):
    total_pages = math.ceil(len(self.all_results) / self.results_per_page)
    if self.current_page < total_pages:
        self.current_page += 1
        self.update_page_display()

def clear_layout(self, layout):
    while layout.count() > 1:
        child = layout.takeAt(0)
        if child.widget():
            child.widget().deleteLater()

def handle_summary_request(self, data: dict):
    self.controller.show_summary_page(data)

def handle_view_cv_request(self, cv_path: str):
    if cv_path and os.path.exists(cv_path):
        QDesktopServices.openUrl(QUrl.fromLocalFile(cv_path))
    else:
        QMessageBox.critical(self, "File Not Found", f"This is a dummy card. The CV file could not be found at:\n{cv_path}")

```

summary_page.py

```

from PyQt6.QtWidgets import (
    QWidget, QVBoxLayout, QHBoxLayout, QLabel,
    QPushButton, QMessageBox, QGroupBox, QFrame, QScrollArea
)
from PyQt6.QtGui import QFont, QDesktopServices
from PyQt6.QtCore import QUrl, Qt
import os

class SummaryPage(QWidget):
    def __init__(self, controller):
        super().__init__()
        self.controller = controller
        self.applicant_data = {}
        self.init_ui()

    def init_ui(self):
        main_layout = QVBoxLayout(self)

        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_area.setObjectName("SummaryScrollArea")

        container_widget = QWidget()

```

```

self.details_layout = QVBoxLayout(container_widget)
self.details_layout.setAlignment(Qt.AlignmentFlag.AlignTop)

scroll_area.setWidget(container_widget)
main_layout.addWidget(scroll_area)

button_layout = QHBoxLayout()
back_button = QPushButton("< Back to Search")
back_button.clicked.connect(self.controller.show_search_page)

view_cv_button = QPushButton("View Original CV")
view_cv_button.clicked.connect(self.on_view_cv)

button_layout.addStretch()
button_layout.addWidget(view_cv_button)
button_layout.addWidget(back_button)
main_layout.addLayout(button_layout)

def _create_section_group(self, title):
    group_box = QGroupBox(title)
    layout = QVBoxLayout(group_box)
    layout.setSpacing(10)
    return group_box, layout

def _create_info_entry(self, title, subtitle="", description=""):
    frame = QFrame()
    frame.setObjectName("InfoEntryFrame")
    layout = QVBoxLayout(frame)
    layout.setContentsMargins(10, 5, 10, 5)

    title_label = QLabel(title)
    title_label.setFont(QFont("Helvetica", 11, QFont.Weight.Bold))
    layout.addWidget(title_label)

    if subtitle:
        subtitle_label = QLabel(subtitle)
        subtitle_label.setObjectName("SubtitleLabel")
        layout.addWidget(subtitle_label)

    if description:
        desc_label = QLabel(description)
        desc_label.setWordWrap(True)
        desc_label.setTextInteractionFlags(Qt.TextInteractionFlag.TextSelectableByMouse)
        layout.addWidget(desc_label)

    return frame

def _clear_layout(self, layout):
    while layout.count():
        child = layout.takeAt(0)
        if child.widget():
            child.widget().deleteLater()

def load_data(self, data):
    print(data)

```



```

self.applicant_data = data
self._clear_layout(self.details_layout)

# --- Identity Section ---
identity_frame = QFrame()
identity_frame.setObjectName("IdentityFrame")
identity_layout = QVBoxLayout(identity_frame)

name_label = QLabel(data.get('name', 'Unknown Applicant'))
name_label.setFont(QFont("Helvetica", 16, QFont.Weight.Bold))
name_label.setObjectName("SummaryName")

details_layout_inner = QVBoxLayout()
details_layout_inner.addWidget(QLabel(f"<b>Birthdate:</b> {data.get('date_of_birth', 'N/A')}"))
details_layout_inner.addWidget(QLabel(f"<b>Address:</b> {data.get('address', 'N/A')}"))
details_layout_inner.addWidget(QLabel(f"<b>Phone:</b> {data.get('phone_number', 'N/A')}"))

identity_layout.addWidget(name_label)
identity_layout.addLayout(details_layout_inner)
self.details_layout.addWidget(identity_frame)

# --- Extracted Info Section ---
summary = data.get('summary', {})

summary_group, summary_layout = self._create_section_group("Summary")

summary_label = QLabel(summary)
summary_label.setWordWrap(True)
summary_layout.addWidget(summary_label)

self.details_layout.addWidget(summary_group)

## --- Skills Section (as plain text) ---
# skills_group, skills_layout = self._create_section_group("Consolidated Skills")
# skills_text = summary.get('Consolidated_Skills', "").strip()
# if skills_text:
#     skills_label = QLabel(skills_text)
#     skills_label.setWordWrap(True)
#     skills_label.setTextInteractionFlags(Qt.TextInteractionFlag.TextSelectableByMouse)
#     skills_layout.addWidget(skills_label)
# else:
#     skills_layout.addWidget(QLabel("No skills found or extracted."))
# self.details_layout.addWidget(skills_group)

## --- Job History Section ---
# history_group, history_layout = self._create_section_group("Job History")
# experience = summary.get('Work_History', [])
# if experience:
#     for item in experience:
#         title = item.get('Company', 'N/A')
#         subtitle = f"({item.get('Date', 'N/A')})"
#         description = item.get('Position', 'N/A')
#         history_layout.addWidget(self._create_info_entry(title, subtitle, description))
# else:
#     history_layout.addWidget(QLabel("No job history found or extracted."))

```

```

# self.details_layout.addWidget(history_group)

# # --- Education Section ---
# education_group, education_layout = self._create_section_group("Education")
# education = summary.get('Education', {})
# if education and education.get('School'):
#     title = education.get('School', 'N/A')
#     subtitle = education.get('Degree', 'N/A')
#     description = f"Years: {education.get('Years', 'N/A')} \n Full Text: {education.get('Full_Text',
'N/A')}"
#     education_layout.addWidget(self._create_info_entry(title, subtitle, description))
# else:
#     education_layout.addWidget(QLabel("No education history found or extracted."))
# self.details_layout.addWidget(education_group)

# # Optional Sections: Summary, Highlights, Accomplishments
# optional_sections = {
#     "Summary": summary.get('Summary'),
#     "Highlights": summary.get('Highlights'),
#     "Accomplishments": summary.get('Accomplishments')
# }

# for section_name, content in optional_sections.items():
#     if content:
#         section_group, section_layout = self._create_section_group(section_name)
#         label = QLabel(content)
#         label.setWordWrap(True)
#         label.setTextInteractionFlags(Qt.TextInteractionFlag.TextSelectableByMouse)
#         section_layout.addWidget(label)
#         self.details_layout.addWidget(section_group)

def on_view_cv(self):
    cv_path = self.applicant_data.get('cv_path')
    if cv_path and os.path.exists(cv_path):
        QDesktopServices.openUrl(QUrl.fromLocalFile(cv_path))
    else:
        QMessageBox.critical(self, "File Not Found", f"The CV file could not be found at: \n {cv_path}")

```

Setelah bagian GUI, terdapat beberapa fungsi helper yang berfungsi untuk berbagai kegunaan termasuk untuk main logic yang berjalan dalam aplikasi

search_worker.py

```

import os
from PyQt6.QtCore import QObject, pyqtSignal
from src.database.query import get_all_applications
from src.extractor.pdf_parser import PDFParser
from src.algorithm.kmp import KMP
from src.algorithm.bm import BM
from src.algorithm.levenshtein import Levenshtein
from src.algorithm.aho_corasick import AhoCorasick
from src.utils.timer import Timer
from src.extractor.regex_extractor import RegexExtractor

```

```

class SearchWorker(QObject):
    search_finished = pyqtSignal(list, float, float)
    search_error = pyqtSignal(str)

    def __init__(self, keywords: list[str], algorithm: str, top_n: int):
        super().__init__()
        self.keywords = keywords
        self.algorithm = algorithm
        self.top_n = top_n
        self.is_cancelled = False

    def run(self):
        timer = Timer()
        try:
            all_cv_tuples = get_all_applications()
        except Exception as e:
            self.search_error.emit(f"Database connection error: {e}")
            return

        timer.start()
        results_dict = {}
        globally_found_keywords = set()

        if self.algorithm == "AhoCorasick":
            ac = AhoCorasick()
            for k in self.keywords:
                ac.add_pattern(k)
            ac.build_failure_links()

        for cv_tuple in all_cv_tuples:
            if self.is_cancelled:
                return

            applicant_id, first_name, last_name, dob, address, phone, cv_path, _ = cv_tuple

            if not cv_path or not os.path.exists(cv_path):
                continue

            raw_text = PDFParser(cv_path).extract_text()
            if not raw_text:
                continue

            search_text = raw_text.lower()
            found_matches = {}

            if self.algorithm == "AhoCorasick":
                matches = ac.search(search_text)
                for _, pattern in matches:
                    original_keyword = next((k for k in self.keywords if k.lower() == pattern), pattern)
                    found_matches[original_keyword] = found_matches.get(original_keyword, 0) + 1
            else:
                for keyword in self.keywords:
                    matcher = KMP(keyword) if self.algorithm == "KMP" else BM(keyword)
                    count = matcher.count_occurrences(search_text)

```

```

        if count > 0:
            found_matches[keyword] = count

    if found_matches:
        globally_found_keywords.update(found_matches.keys())
        regex = RegexExtractor(raw_text)
        summary_data = regex.extract_all()
        results_dict[applicant_id] = {
            'applicant_id': applicant_id,
            'name': f'{first_name} {last_name}'.strip(),
            'date_of_birth': dob,
            'address': address,
            'phone_number': phone,
            'cv_path': cv_path,
            'matched_keywords': found_matches,
            'summary': summary_data
        }

    timer.stop()
    exact_time_ms = timer.elapsed_ms()

    unfound_keywords = [k for k in self.keywords if k not in globally_found_keywords]
    fuzzy_time_ms = 0.0

    if unfound_keywords:
        timer.start()
        fuzzy_matcher = Levenshtein(threshold=0.85)

        for cv_tuple in all_cv_tuples:
            if self.is_cancelled:
                return

            applicant_id, first_name, last_name, dob, address, phone, cv_path, _ = cv_tuple
            res_entry = results_dict.get(applicant_id)

            if res_entry:
                continue

            if not cv_path or not os.path.exists(cv_path):
                continue

            raw_text = PDFParser(cv_path).extract_text()
            if not raw_text:
                continue

            similar_found = fuzzy_matcher.find_similar_keywords(unfound_keywords, raw_text)

            if similar_found:
                summary_data = RegexExtractor(raw_text).extract_all()
                res_entry = {
                    'applicant_id': applicant_id,
                    'name': f'{first_name} {last_name}'.strip(),
                    'date_of_birth': dob,
                    'address': address,
                    'phone_number': phone,

```

```

        'cv_path': cv_path,
        'matched_keywords': {},
        'summary': summary_data
    }
    results_dict[applicant_id] = res_entry

    for keyword, matches in similar_found.items():
        res_entry['matched_keywords'][f"{keyword} (fuzzy)"] = len(matches)

    timer.stop()
    fuzzy_time_ms = timer.elapsed_ms()

    final_results = list(results_dict.values())
    for res in final_results:
        res['match_count'] = len(res['matched_keywords'])

    final_results.sort(key=lambda x: x.get('match_count', 0), reverse=True)

    self.search_finished.emit(final_results[:self.top_n], exact_time_ms, fuzzy_time_ms)

    def cancel(self):
        self.is_cancelled = True

```

file_utils.py

```

import os

def is_pdf_file(path: str) -> bool:
    """Memastikan file berformat PDF dan eksis"""
    return os.path.isfile(path) and path.lower().endswith(".pdf")

def get_filename(path: str) -> str:
    """Mendapatkan nama file dari path"""
    return os.path.basename(path)

```

[similarity.py](#)

```

from src.algorithm.levenshtein import *

def fuzzy_match_keywords(keywords: list[str], text: str, threshold: float = 0.8) -> dict[str, list[str]]:
    """
    Untuk setiap keyword, cari kata dalam teks yang mirip.
    Return dict keyword → list kata dalam teks yang mirip.
    """
    words_in_text = set(text.lower().split())
    matcher = Levenshtein(threshold)
    result = {}

    for keyword in keywords:
        keyword_lower = keyword.lower()
        result[keyword] = [
            word for word in words_in_text if matcher.is_similar(keyword_lower, word)
        ]

```

```
return result
```

timer.py

```
import time

class Timer:
    def __init__(self):
        self.start_time = 0
        self.end_time = 0

    def start(self):
        self.start_time = time.time()

    def stop(self):
        self.end_time = time.time()

    def elapsed_ms(self) -> float:
        """Mengembalikan waktu dalam milidetik"""
        return (self.end_time - self.start_time) * 1000
```

Untuk penggunaan database, database yang digunakan adalah mysql yang diinstantiasi menggunakan sebuah docker dan versi python yang digunakan adalah versi 3.13. Berikut merupakan program [main.py](#) sebagai program utama

main.py

```
import os
import sys
from PyQt6.QtCore import QDir
from PyQt6.QtWidgets import QApplication
from src.gui.main_window import MainWindow
import falthandler

falthandler.enable()

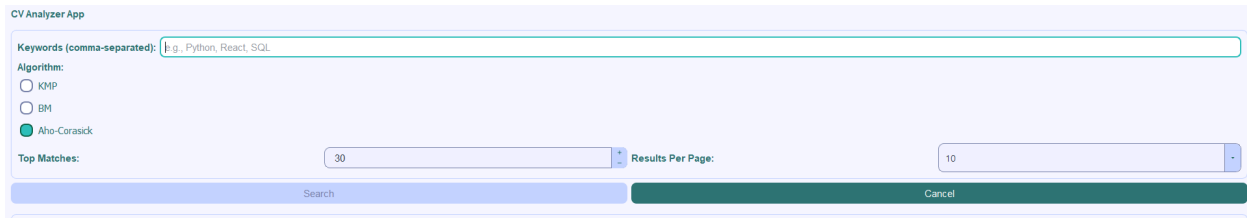
if __name__ == "__main__":
    root = os.path.dirname(os.path.abspath(__file__))
    QDir.addSearchPath('icon', os.path.join(root, 'icon'))
    app = QApplication(sys.argv)

    # Load Stylesheet
    try:
        with open("style.css", "r") as f:
            style = f.read()
            app.setStyleSheet(style)
            print("Stylesheet loaded successfully.")
    except FileNotFoundError:
        print("Stylesheet file 'style.qss' not found. Using default styles.")
```

```
main_win = MainWindow()
main_win.show()
sys.exit(app.exec())
```

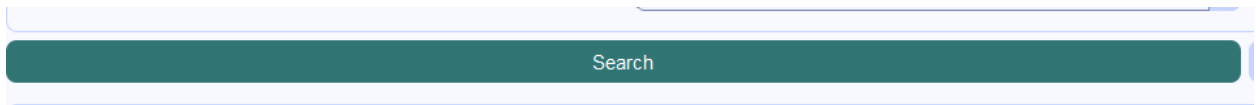
4.2. Penjelasan Tata Cara Penggunaan Program

1. Masukkan keyword yang diinginkan, algoritma yang ingin digunakan, dan jumlah hasil yang ingin dikeluarkan



The screenshot shows the 'CV Analyzer App' search interface. It features a text input field for 'Keywords (comma-separated):' containing 'I-g., Python, React, SQL'. Below this, there are radio buttons for 'Algorithm:' with options 'KMP', 'BM', and 'Aho-Corasick' (which is selected). To the right, there are two numeric input fields: 'Top Matches:' set to '30' and 'Results Per Page:' set to '10'. At the bottom, there are 'Search' and 'Cancel' buttons.

2. Klik tombol “Search” dan tunggu hasilnya

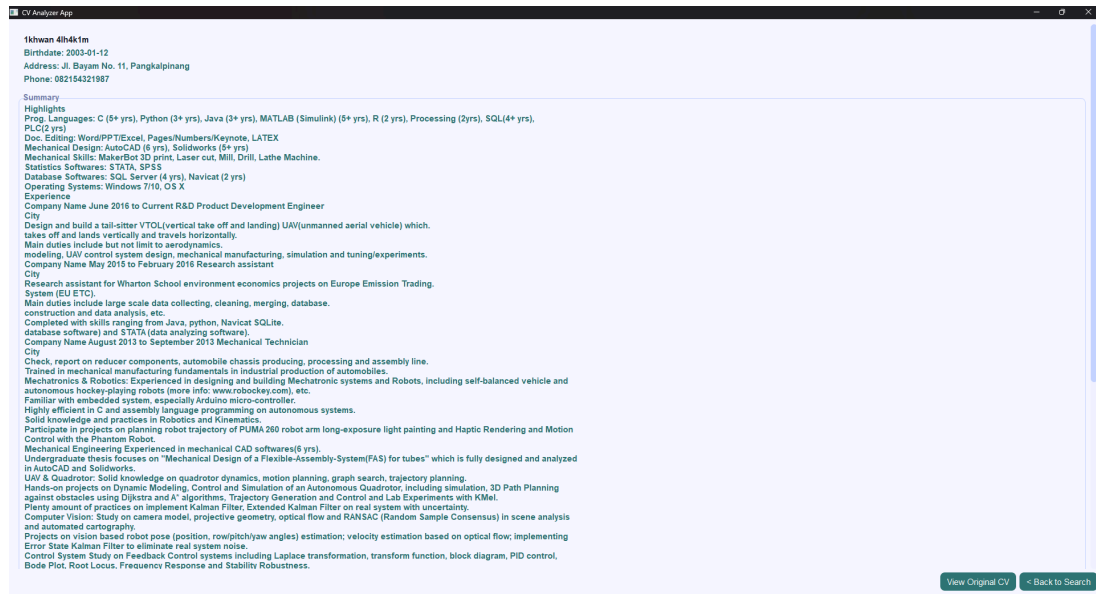


3. Program akan menghasilkan jumlah hasil sesuai dengan jumlah permintaan, jika kurang akan dilakukan fuzzy match

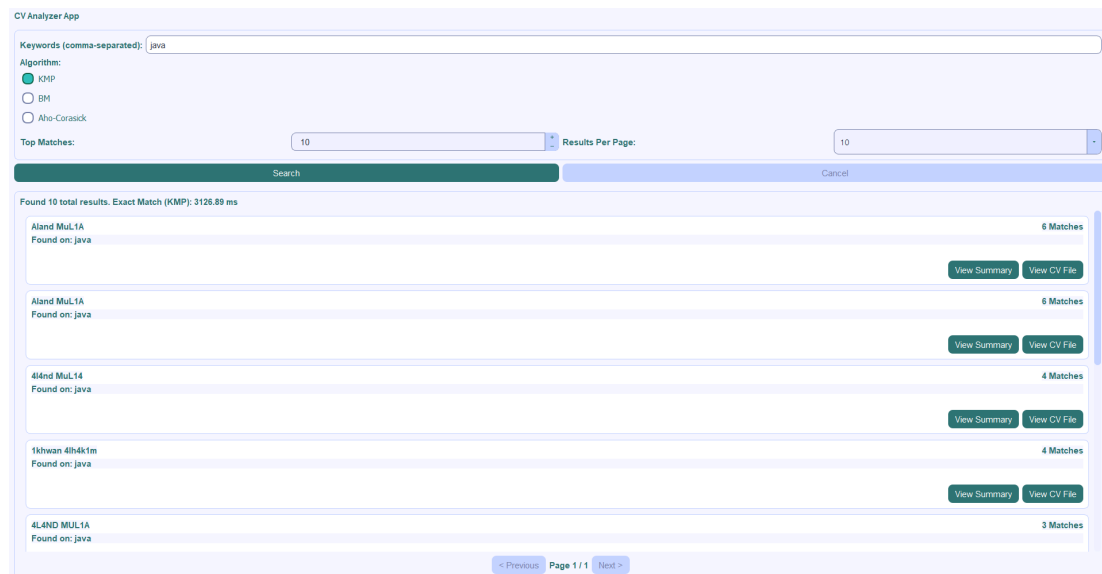


The screenshot displays the search results page. At the top, it states 'Found 30 total results. Exact Match (AhoCorasick): 3194.51 ms | Fuzzy Match: 5386.95 ms'. Below this, there are five result entries, each showing a name, a 'Found on:' field, and a 'Matches' count. The first entry is '1khwan 4lh4k1m' with 4 matches. The next three entries are 'MOHAMMAD NUGRAHA' with 4 matches each. The last entry is 'M0h4mM4D NUGRAHA' with 3 matches. Each entry has 'View Summary' and 'View CV File' buttons. At the bottom, there are navigation links: '< Previous', 'Page 1 / 3', and 'Next >'.

4. Anda bisa melihat summary dan PDF aslinya, PDF aslinya akan terbuka di eksternal



4.3. Hasil Pengujian



CV Analyzer App

Keywords (comma-separated):

Algorithm:

☐ KMP

☒ BM

☐ Aho-Corasic

Top Matches: Results Per Page:

[Search](#) [Cancel](#)

Found 10 total results. Exact Match (BM): 2872.55 ms

Aland Mul1A Found on: java	6 Matches	View Summary View CV File
Aland Mul1A Found on: java	6 Matches	View Summary View CV File
4land Mul14 Found on: java	4 Matches	View Summary View CV File
1khwan 4h4k1m Found on: java	4 Matches	View Summary View CV File
4LAND MUL1A Found on: java	3 Matches	

[< Previous](#) [Page 1 / 1](#) [Next >](#)

CV Analyzer App

Keywords (comma-separated):

Algorithm:

☐ KMP

☐ BM

☒ Aho-Corasic

Top Matches: Results Per Page:

[Search](#) [Cancel](#)

Found 10 total results. Exact Match (AhoCorasick): 2976.79 ms

Aland Mul1A Found on: java	6 Matches	View Summary View CV File
Aland MUL1A Found on: java	6 Matches	View Summary View CV File
4land Mul14 Found on: java	4 Matches	View Summary View CV File
1khwan 4h4k1m Found on: java	4 Matches	View Summary View CV File
4LAND MUL1A Found on: java	3 Matches	

[< Previous](#) [Page 1 / 1](#) [Next >](#)

CV Analyzer App

Keywords (comma-separated):

Algorithm:

☒ KMP

☐ BM

☐ Aho-Corasick

Top Matches: Results Per Page:

[Search](#) [Cancel](#)

Found 30 total results. Exact Match (KMP): 3564.35 ms

1khwan 4ih4k1m Found on: java, python	8 Matches
View Summary View CV File	
Aland MuL1A Found on: java	6 Matches
View Summary View CV File	
Aland MuL1A Found on: java	6 Matches
View Summary View CV File	
H41k4l 455y4uq1 Found on: java, python	4 Matches
View Summary View CV File	
H41k4l 455y4uq1 Found on: java, python	4 Matches
View Summary View CV File	

[< Previous](#) [Page 1 / 3](#) [Next >](#)

CV Analyzer App

Keywords (comma-separated):

Algorithm:

☐ KMP

☒ BM

☐ Aho-Corasick

Top Matches: Results Per Page:

[Search](#) [Cancel](#)

Found 30 total results. Exact Match (BM): 3285.15 ms

1khwan 4ih4k1m Found on: java, python	8 Matches
View Summary View CV File	
Aland MuL1A Found on: java	6 Matches
View Summary View CV File	
Aland MuL1A Found on: java	6 Matches
View Summary View CV File	
H41k4l 455y4uq1 Found on: java, python	4 Matches
View Summary View CV File	
H41k4l 455y4uq1 Found on: java, python	4 Matches
View Summary View CV File	

[< Previous](#) [Page 1 / 3](#) [Next >](#)

CV Analyzer App

Keywords (comma-separated):

Algorithm:

☐ KMP

☐ BM

☒ Aho-Corasic

Top Matches: Results Per Page:

[Search](#) [Cancel](#)

Found 30 total results. Exact Match (AhoCorasick): 3295.86 ms

1khwan 4th4k1m Found on: java, python	8 Matches	View Summary View CV File
Alamd Mul.1A Found on: java	6 Matches	View Summary View CV File
Alamd Mul.1A Found on: java	6 Matches	View Summary View CV File
H41k4l 455y4uq1 Found on: java, python	4 Matches	View Summary View CV File
H41k4l 455y4uq1 Found on: java, python	4 Matches	View Summary View CV File

[< Previous](#) [Page 1 / 3](#) [Next >](#)

CV Analyzer App

Keywords (comma-separated):

Algorithm:

☐ KMP

☐ BM

☒ Aho-Corasic

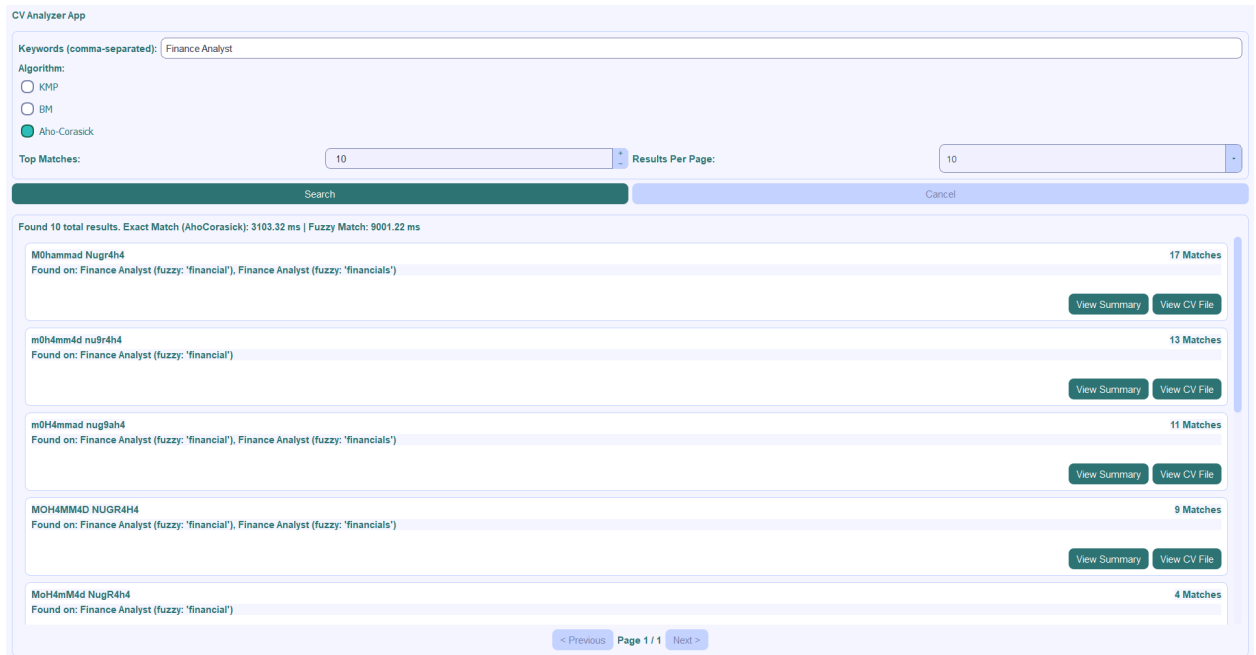
Top Matches: Results Per Page:

[Search](#) [Cancel](#)

Found 50 total results. Exact Match (AhoCorasick): 3155.39 ms | Fuzzy Match: 5719.67 ms

MoH4mM4d NugR4h4 Found on: python (fuzzy: 'position'), python (fuzzy: 'within')	7 Matches	View Summary View CV File
MoH4mM4d NugR4h4 Found on: python (fuzzy: 'action'), python (fuzzy: 'within')	7 Matches	View Summary View CV File
MoH4mM4d NugR4h4 Found on: python (fuzzy: 'position'), python (fuzzy: 'within')	7 Matches	View Summary View CV File
M0hammad Nugr4h4 Found on: python (fuzzy: 'position'), python (fuzzy: 'within')	6 Matches	View Summary View CV File
1khwan 4th4k1m Found on: python	4 Matches	View Summary View CV File

[< Previous](#) [Page 1 / 5](#) [Next >](#)



4.4. Analisis Hasil

Berdasarkan serangkaian pengujian yang divisualisasikan dalam gambar-gambar yang diberikan, aplikasi CV Analyzer menunjukkan fungsionalitas yang kuat dan sesuai dengan tujuan utamanya, yaitu untuk menyaring dan menganalisis CV kandidat secara efisien. Pengujian ini mencakup berbagai skenario pencarian dengan kata kunci tunggal, kata kunci ganda, serta perbandingan kinerja antar algoritma pencocokan string.

Pada pengujian dengan kata kunci ganda seperti "java, python" menggunakan algoritma Aho-Corasick, aplikasi berhasil menemukan 30 kandidat yang relevan. Hasilnya ditampilkan dalam format *card* yang jelas, mencantumkan nama kandidat dan jumlah total kata kunci yang cocok (contohnya, "8 Matches"). Ini membuktikan bahwa sistem pemeringkatan berdasarkan jumlah kecocokan berfungsi dengan baik, menempatkan kandidat yang paling relevan di urutan teratas. Fungsionalitas paginasi (halaman) juga berjalan lancar, memungkinkan pengguna untuk menavigasi daftar hasil yang panjang dengan mudah.

Pengujian yang membandingkan algoritma KMP, Boyer-Moore (BM), dan Aho-Corasick menunjukkan bahwa ketiganya berhasil diimplementasikan dan memberikan hasil yang akurat. Misalnya, saat mencari kata kunci "java", baik KMP maupun BM mampu menemukan 10 hasil teratas dengan benar. Aplikasi ini juga mencatat dan menampilkan waktu eksekusi untuk setiap pencarian, seperti "Exact Match (KMP): 3126.88 ms". Fitur ini sangat berharga untuk analisis perbandingan kinerja di masa depan, memberikan data konkret tentang efisiensi masing-masing algoritma dalam berbagai kondisi.

Salah satu fitur unggulan yang teruji adalah kemampuan pencarian fuzzy. Ketika kata kunci "Finance Analyst" digunakan, aplikasi tidak hanya menemukan kecocokan yang persis tetapi juga variasi yang mirip secara fonetis atau ejaan, seperti "financial" dan "financials". Hasilnya dengan jelas melabeli temuan ini sebagai fuzzy (contoh: "Finance Analyst (fuzzy: 'financial')"), yang menunjukkan transparansi dan kecerdasan sistem. Kemampuan untuk secara otomatis memperluas pencarian dengan metode fuzzy ketika hasil pencarian eksak kurang dari yang diminta (misalnya, meminta 10 hasil teratas) memastikan bahwa pengguna selalu mendapatkan jumlah kandidat yang relevan dan memadai.

Saat pengguna memilih seorang kandidat dari daftar hasil, aplikasi dengan sukses menampilkan halaman ringkasan yang terperinci. Seperti yang terlihat pada kasus kandidat "Ikhwan Alhakim", aplikasi berhasil mengekstrak dan menyajikan informasi krusial dari CV, termasuk data pribadi, ringkasan keahlian (Highlights), riwayat pekerjaan (Experience), dan riwayat pendidikan (Education). Kemampuan untuk beralih antara ringkasan yang diekstrak secara otomatis dan melihat file CV asli memberikan fleksibilitas penuh kepada perekrut untuk melakukan analisis mendalam.

Secara keseluruhan, hasil pengujian ini mengonfirmasi bahwa aplikasi CV Analyzer adalah alat yang andal, efisien, dan kaya fitur. Aplikasi ini berhasil mengintegrasikan beberapa algoritma pencarian canggih dengan antarmuka pengguna yang intuitif, menjadikannya solusi yang sangat efektif untuk otomatisasi proses penyaringan CV.

Algoritma KMP dan BM memiliki perbedaan pada ketentuan yang sama, algoritma KMP membutuhkan waktu yang lebih lama dibanding BM karena pergeseran string memang lebih cepat dalam BM. Hanya saja ada kesalahan pada algoritma Aho-Corasick karena algoritma yang seharusnya lebih cepat dibanding BM, tetapi menjadi lebih lambat.

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

A. Kesimpulan

Program berhasil berjalan, implementasi algoritma KMP dan BM dapat beroperasi sesuai aturan dan definisi masing-masing dalam melakukan string matching berbasis keyword pada informasi yang ada pada CV berhasil, dengan masing-masing algoritma dapat memiliki kasus masing-masing dalam pengujian kasus tertentu.

B. Saran

Tidak ada saran yang diberikan.

C. Refleksi

Mungkin jika dimulai lebih awal dan/atau ada waktu yang berlebih, hardcode untuk operasi yang memanfaatkan regex bisa mencakup beberapa informasi lain yang mungkin bisa mendapat banyak informasi yang mungkin berkaitan lainnya.

LAMPIRAN

Tabel Implementasi Program:

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 <i>Levenshtein Distance</i> 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 <i>Aho-Corasick</i> .	✓	
11	Membuat bonus video dan diunggah pada Youtube.		✓

Link Github Repository: https://github.com/filbertengyo/Tubes3_U_Tiga

DAFTAR PUSTAKA

<https://pdfs.semanticscholar.org/1809/639e04c201682b64708c3b500af29517d2ed.pdf>
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Makalah/Makalah-IF2211-Stima-2022-K3%20\(42\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Makalah/Makalah-IF2211-Stima-2022-K3%20(42).pdf)