

LAPORAN TUGAS BESAR 3

IF2211 STRATEGI ALGORITMA

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




Disusun Oleh:

“micin kriuk”

Aloisius Adrian Stevan Gunawan	13523054
Albertus Christian Poandy	13523077
Michael Alexander Angkawijaya	13523102

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

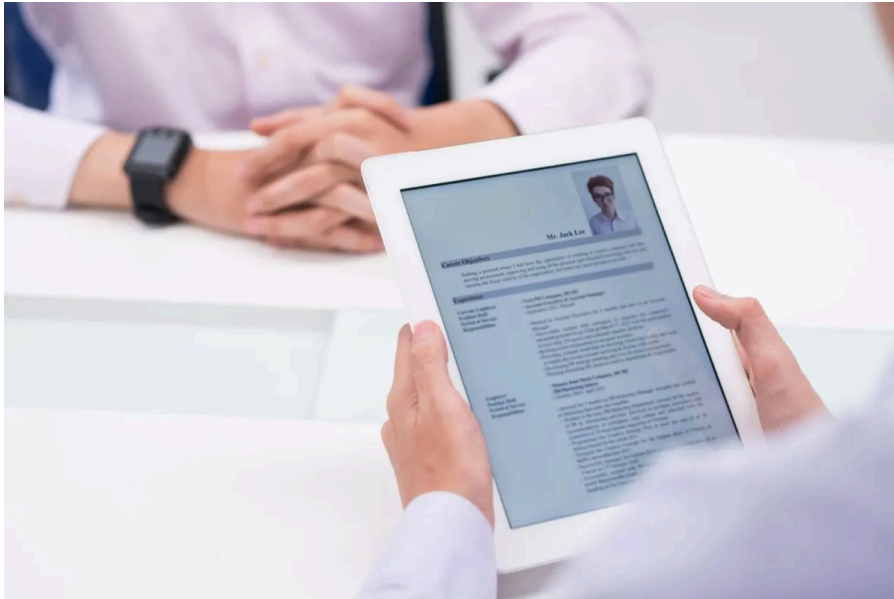
DAFTAR ISI	1
DAFTAR GAMBAR	2
BAB 1	3
DESKRIPSI TUGAS	3
BAB 2	7
LANDASAN TEORI	7
2.1 Algoritma Knuth-Morris-Pratt	7
2.2 Algoritma Boyer-Moore	10
2.3 Regex	10
2.4 Aho-Corasick	11
BAB 3	13
ANALISIS PEMECAHAN MASALAH	13
3.1 Langkah Pemecahan Masalah	13
3.2 Pemetaan masalah menjadi elemen-elemen algoritma String Matching	14
3.3 Fitur Fungsional dan Arsitektur Aplikasi	14
3.3.1 Antarmuka Pengguna (GUI)	14
3.3.2 Mesin Pencarian	15
3.3.3 Basis Data dan Penyimpanan	15
3.4 Ilustrasi Penggunaan	15
BAB 4	17
IMPLEMENTASI DAN PENGUJIAN	17
BAB 5	18
KESIMPULAN, SARAN, DAN REFLEKSI	18
5.1 Kesimpulan	18
5.2 Saran	18
5.3 Refleksi	18
LAMPIRAN	19
Tautan Repository Github	19
Hasil Akhir Tugas Besar	19
	19
DAFTAR PUSTAKA	20

DAFTAR GAMBAR

Gambar 1. CV ATS dalam Dunia Kerja (Sumber: https://www.antaranews.com/)	3
Gambar 2.3.1. Notasi Regex secara umum (Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf)	10
Gambar 2.3.2. Contoh penggunaan regex pada Python (Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf)	11

BAB 1

DESKRIPSI TUGAS



Gambar 1. CV ATS dalam Dunia Kerja (Sumber: <https://www.antaranews.com/>)

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

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

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

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

1. Sistem yang dibangun pada tugas besar ini bertujuan untuk melakukan pencocokan dan pencarian data pelamar kerja berbasis CV yang diunggah oleh pengguna. Sistem dikembangkan menggunakan **bahasa pemrograman Python** dengan **antarmuka desktop** berbasis pustaka seperti **Tkinter, PyQt, atau framework lain** yang relevan. Dalam proses pencocokan kata kunci, sistem wajib mengimplementasikan algoritma **Knuth-Morris-Pratt (KMP)** dan **Boyer-Moore (BM)**. Untuk mengukur kemiripan saat terjadi kesalahan input atau perbedaan penulisan, sistem juga menerapkan algoritma **Levenshtein Distance**. Selain itu, **Regular Expression (Regex)** digunakan untuk mengekstrak informasi penting dari teks CV secara otomatis. Oleh karena itu, penguasaan pengembangan GUI dan pemrosesan string di Python sangat penting untuk menyelesaikan tugas ini secara optimal.
2. Program yang dikembangkan harus menggunakan basis data berbasis **MySQL** untuk menyimpan informasi hasil ekstraksi dari CV yang telah diunggah. Basis data akan menyimpan informasi berupa profil pelamar beserta lokasi penyimpanan file CV di dalam sistem.
3. Fitur utama dari sistem ini adalah kemampuannya untuk ekstraksi teks dari CV dalam format PDF. Setelah dokumen CV diunggah, program harus mampu melakukan ekstraksi teks secara otomatis dan mengubahnya menjadi profil pelamar kerja. Profil tersebut akan ditampilkan kepada pengguna tanpa perlu intervensi manual tambahan. Proses ini akan membantu mempercepat identifikasi dan penilaian awal terhadap pelamar.
4. Sistem wajib menyediakan fitur **pencarian terhadap data pelamar** menggunakan **kata kunci** atau kriteria tertentu yang ditentukan oleh pengguna (misalnya nama, skill tertentu, atau pengalaman kerja). Pencarian ini akan dilakukan terhadap semua data dalam database dan bertujuan untuk menemukan pelamar yang paling relevan dengan kriteria pencarian tersebut. Proses pencarian dilakukan sepenuhnya secara in-memory agar hasilnya cepat dan responsif. Proses pencarian utamanya dilakukan secara *exact matching*.
5. Setelah *exact matching*, apabila tidak ditemukan kecocokan secara persis, sistem harus melakukan *fuzzy matching*. Untuk setiap kata kunci yang tidak ditemukan satupun kemunculan saat *exact matching*, lakukan pencarian kembali dengan **perhitungan**

tingkat kemiripan menggunakan algoritma Levenshtein Distance. Algoritma ini memungkinkan sistem untuk tetap menampilkan hasil pencarian yang relevan, meskipun terdapat perbedaan minor atau kesalahan ketik pada input pengguna. Hal ini sangat membantu pengguna untuk tetap mendapatkan hasil terbaik tanpa harus memasukkan kata kunci secara sempurna.

6. Apabila salah satu hasil pencarian di-klik, sistem harus dapat menampilkan **ringkasan/summary** dari lamaran tersebut. Pada halaman ringkasan, harus terdapat opsi (e.g. tombol) untuk **melihat CV secara keseluruhan**.
7. Informasi yang ditampilkan dalam **ringkasan/summary** CV dari hasil pencarian harus mencakup **data penting dari pelamar**, yaitu identitas (nama, kontak, dan informasi pribadi lainnya) yang diperoleh dari basis data. Kemudian terdapat beberapa data yang diperoleh dengan cara ekstraksi melalui **regular expression**, meliputi:
 - Ringkasan pelamar (summary/overview)
 - Keahlian pelamar (skill)
 - Pengalaman kerja (e.g. tanggal dan jabatan)
 - Riwayat pendidikan (e.g. tanggal kelulusan, universitas, dan gelar)Dengan menampilkan informasi-informasi penting tersebut, sistem dapat memberikan ringkasan profil yang relevan kepada pengguna.
8. Pengguna aplikasi dapat memilih algoritma pencocokan yang ingin digunakan untuk *exact matching*, yaitu antara **KMP** atau **BM**, (bisa pula **Aho-Corasick** apabila mengerjakan **bonus**), sebelum memulai proses pencarian. Pilihan algoritma ini akan mempengaruhi cara sistem memindai dan mencocokkan kata kunci dengan isi CV. Hal ini memberikan fleksibilitas dan pemahaman algoritmik yang lebih luas bagi pengguna atau pengembang.
9. Pengguna aplikasi dapat **menentukan jumlah CV yang ditampilkan**. CV yang ditampilkan diurutkan mulai dari CV dengan jumlah kecocokan kata kunci terbanyak.
10. Setelah pencarian CV, sistem akan **menampilkan waktu pencarian**. Terdapat **2 waktu berbeda** yang perlu ditampilkan. Pertama adalah waktu pencarian **exact match** menggunakan algoritma KMP atau BM. Kemudian, tampilkan juga waktu pencarian **fuzzy match** menggunakan Levenshtein Distance apabila terdapat kata kunci yang belum ditemukan.
11. Aplikasi yang dibuat harus memiliki **antarmuka pengguna (user interface) yang intuitif dan menarik**, sehingga mudah digunakan bahkan oleh pengguna awam. Komponen-komponen penting seperti, input *keyword*, pemilihan algoritma, serta hasil

pencarian harus disusun dengan jelas dan rapi. Pengembang juga diperkenankan menambahkan fitur tambahan yang dapat memperkaya fungsi dan pengalaman pengguna, sebagai bentuk kreativitas dan inisiatif dalam mengembangkan sistem yang lebih bermanfaat dan inovatif.

BAB 2

LANDASAN TEORI

2.1 Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) adalah salah satu algoritma pencarian *substring* dari sebuah *string* yang lebih panjang atau sama dengan *substring* tersebut. Teknik pencarian ini mirip dengan algoritma *brute force*, yaitu dilakukan pencarian dari kiri ke kanan. Namun, algoritma ini memiliki alur pencarian yang berbeda dari algoritma *brute force*.

Pada algoritma *brute force*, saat ditemukan karakter yang tidak sama pada *substring* tersebut, “jendela” pada string akan digeser satu. Proses ini dapat menjadi redundan, dan bisa menghambat kinerja dari algoritma *brute force*. Algoritma KMP akan mengurangi pencarian yang redundan ini. Dengan melakukan *preprocessing*, algoritma ini dapat mengurangi redundansi yang terjadi saat melakukan pencocokan *string*.

Teknik *preprocessing* yang dilakukan pada awal pencarian akan menyimpan nilai khusus dari *substring* yang ingin dicari. Misalkan diberikan sebuah substring yang memiliki panjang n . Untuk setiap karakter pada *substring* yang ingin dicari, akan dimasukkan indeks *substring* tersebut agar tidak terjadi redundansi. Berikut adalah *pseudocode* dari pengaplikasian algoritma *preprocessing* KMP:

```
FUNCTION computeLPSArray(pattern, M):
    CREATE an integer array lps of size M
    length = 0
    lps[0] = 0
    i = 1
    WHILE i < M:
        IF pattern[i] == pattern[length]:
            length = length + 1
            lps[i] = length i = i + 1
        ELSE (pattern[i] != pattern[length]):
            IF length != 0: length = lps[length - 1]
            ELSE (length == 0): lps[i] = 0 i = i + 1
    RETURN lps
```

Setelah melakukan *preprocessing*, pencarian akan dilakukan mirip dengan algoritma *brute force*, namun setiap kali terdapat *mismatch* atau ketidakcocokan, “jendela” akan melompat ke index substring berikutnya yang kondisi prefixnya masih terpenuhi, sehingga redundansi dapat terhindarkan. Berikut adalah *pseudocode* untuk algoritma KMP:


```

FUNCTION KMPSearch(text, pattern):
    N = length of text
    M = length of pattern
    lps = computeLPSArray(pattern, M)
    i = 0 // index for text[]
    j = 0 // index for pattern[]
    WHILE i < N: IF pattern[j] == text[i]:
        i = i + 1
        j = j + 1
        IF j == M: PRINT "Pattern found at index " + (i - j)
        j = lps[j - 1]
    ELSE IF i < N AND pattern[j] != text[i]:
        IF j != 0:
            j = lps[j - 1]
        ELSE:
            i = i + 1

```

Misalkan terdapat sebuah *substring* ABABA. Berikut adalah contoh simulasi *preprocessing* string tersebut:

ABABA 00000 ↑	Huruf A pada indeks 0 muncul untuk pertama kalinya.
ABABA 00000 ↑	Huruf B pada indeks 1 muncul untuk pertama kalinya.
ABABA 00100 ↑	Huruf A pada indeks 2 sama dengan huruf A pada indeks 0, maka diberikan nilai 1.

ABABA 00120 ↑	Huruf B pada indeks 3 sama dengan huruf B pada indeks prefix sekarang + 1 (indeks 1), maka diberikan nilai 2.
ABABA 00123 ↑	Huruf A pada indeks 4 sama dengan huruf A pada indeks prefix sekarang + 1 (indeks 2), maka diberikan nilai 3.

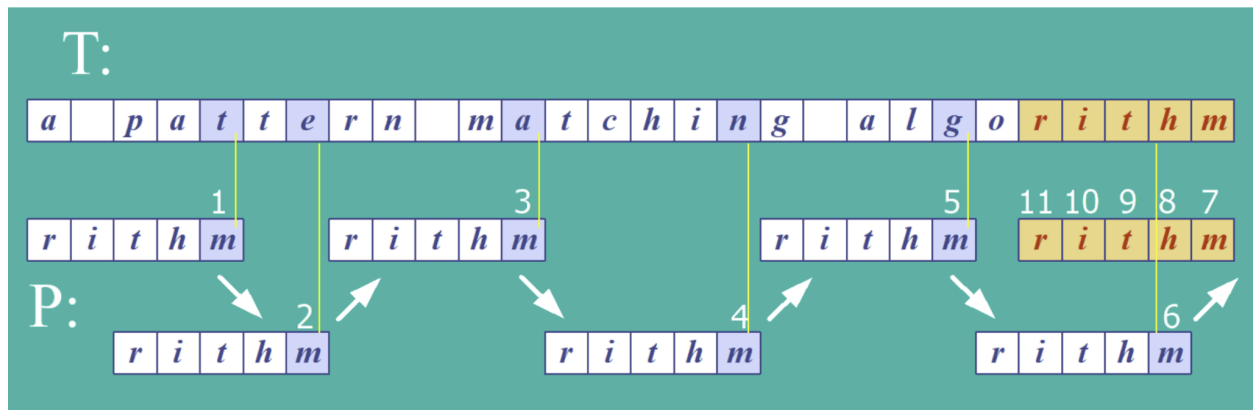
Setelah mendapatkan larik *lps*, kita dapat melakukan pencarian dengan lebih mudah. Misalkan diberikan sebuah *string* ABAABABA, dan ingin dicari *substring* ABABA. Berikut adalah simulasi pencarian *substring*-nya:

Huruf string (index i)	i	Huruf substring (index j)	j	verdict
A	0	A	0	COCOK
B	1	B	1	COCOK
A	2	A	2	COCOK
A	3	B	3	TIDAK COCOK, sehingga mundur ke index $lsp[j-1]$, dan $lsp[2] = 1$
A	3	B	1	TIDAK COCOK, sehingga mundur ke index $lsp[j-1]$, dan $lsp[0] = 0$
A	4	A	0	COCOK
B	5	B	1	COCOK
A	6	A	2	COCOK
B	7	B	3	COCOK
A	8	A	4	COCOK (DITEMUKAN)

2.2 Algoritma Boyer-Moore

Algoritma Boyer-Moore juga merupakan algoritma pencarian *substring* pada sebuah string atau teks tertentu. Algoritma ini juga melakukan *preprocessing* untuk membantu keberjalanan algoritmanya. Berbeda dengan algoritma KMP, pengecekan string dilakukan dari karakter paling terakhir dari sebuah pola yang ingin dicari.

Teknik yang digunakan pada algoritma Boyer-Moore berpusat pada dua teknik, yaitu teknik *looking-glass* dan teknik *character-jump*. *Looking-glass* technique adalah metode pengecekan string secara mundur, dimulai dari karakter dari paling belakang *string*. Sedangkan, teknik *character-jump* memiliki kompleksitas yang sedikit lebih tinggi. Teknik ini akan digunakan ketika terjadi pencocokan karakter yang tidak sama.



Gambar 2.2.1. Contoh algoritma Boyer-Moore (Sumber:

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf))

Pada gambar 2.2.1, dapat dilihat bahwa terdapat beberapa pencocokan karakter yang berbeda pada contoh tersebut. Teknik *character-jump* akan menentukan berapa lompatan yang diperlukan ketika terdapat ketidakcocokan. Pada kasus pertama, seperti pada langkah-1 ke langkah-2, *Pattern* bergeser dua ke kanan, karena terdapat pada saat mencocokkan *Text*, huruf 't' terdapat pada *pattern*. Kasus yang lain terjadi adalah dari langkah ke-2 ke langkah ke-3, dimana tidak ada huruf 'e' pada *pattern*, sehingga langsung digeser penuh.

2.3 Regex

Regular Expression adalah suatu kumpulan ekspresi menggunakan karakter yang digunakan untuk mencari sebuah pola dari kumpulan teks. Pada pengembangan aplikasi berbasis web ini, *regex* digunakan untuk mencari tahu beberapa informasi penting seperti edukasi, dan pengalaman dari pendaftar.

.	Any character except newline.
\.	A period (and so on for *, \ (, \\, etc.)
^	The start of the string.
\$	The end of the string.
\d,\w,\s	A digit, word character [A-Za-z0-9_], or whitespace.
\D,\W,\S	Anything except a digit, word character, or whitespace.
[abc]	Character a, b, or c.
[a-z]	a through z.
[^abc]	Any character except a, b, or c.
aa bb	Either aa or bb.
?	Zero or one of the preceding element.
*	Zero or more of the preceding element.
+	One or more of the preceding element.
{n}	Exactly <i>n</i> of the preceding element.
{n,}	<i>n</i> or more of the preceding element.
{m,n}	Between <i>m</i> and <i>n</i> of the preceding element.
??,*?,+?, {n}?, etc.	Same as above, but as few as possible.
(expr)	Capture <i>expr</i> for use with \1, etc.
(?:expr)	Non-capturing group.
(?=expr)	Followed by <i>expr</i> .
(?!expr)	Not followed by <i>expr</i> .

Gambar 2.3.1. Notasi *Regex* secara umum (Sumber:

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf))

Pada umumnya, notasi regex mengikuti pola pada gambar 2.3.1. Pada python, penggunaan regex adalah dengan meng-enskapsulasi notasi regex dengan tanda petik dua, diawali r. Seperti contoh, pada gambar 2.3.2, program akan mencari pola pada teks yang memiliki 4 digit berurutan.

```
pattern = re.compile(r"(\d{4})")
```

Gambar 2.3.2. Contoh penggunaan *regex* pada Python (Sumber:

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf))

2.4 Aho-Corasick

Algoritma Aho-Corasick adalah algoritma pencocokan pola (*pattern matching*) yang dirancang untuk menemukan banyak pola sekaligus dalam sebuah teks secara efisien. Berbeda dengan algoritma pencocokan string lain seperti Knuth-Morris-Pratt (KMP) atau Boyer-Moore yang hanya menangani satu pola dalam satu waktu, Aho-Corasick mampu mencari semua kemunculan dari sekumpulan kata kunci (*keywords*) dalam satu kali pemrosesan teks. Algoritma

ini sangat cocok digunakan untuk aplikasi seperti pencarian kata sensitif, *spam filtering*, atau *auto-complete*, yang membutuhkan pencarian cepat terhadap banyak pola sekaligus.

Langkah awal dari algoritma ini adalah melakukan *preprocessing* terhadap seluruh daftar kata kunci yang ingin dicari. Kata-kata ini akan dimasukkan ke dalam sebuah struktur data pohon yang disebut *trie*, di mana setiap karakter dari suatu kata kunci menjadi simpul dalam pohon tersebut. Setelah *trie* terbentuk, algoritma membangun *failure link* (tangkai kegagalan) antar simpul, yang berfungsi sebagai jalur alternatif saat pencocokan gagal. Proses ini menggunakan algoritma Breadth-First Search (BFS) untuk menelusuri simpul demi simpul dan menghubungkannya dengan simpul *fallback* yang memiliki awalan yang sama.

Pada saat melakukan pencarian dalam teks, Aho-Corasick akan memanfaatkan *trie* dan *failure link* tersebut untuk menelusuri karakter demi karakter dari teks secara linear, tanpa perlu mengulang kembali pencocokan dari awal. Ketika pencocokan karakter gagal, algoritma cukup mengikuti *failure link* ke simpul lain yang relevan, tanpa harus kembali ke akar *trie*. Proses ini mirip seperti *shortcut* yang mempercepat pencarian, membuat algoritma ini sangat efisien. Setiap kali mencapai simpul yang menandakan akhir dari suatu pola, algoritma akan mencatat bahwa pola tersebut telah ditemukan pada posisi tersebut dalam teks.

Salah satu keunggulan utama dari Aho-Corasick adalah kompleksitas waktu yang optimal, yaitu $O(n + m + z)$, di mana n adalah panjang teks, m adalah jumlah total karakter dari semua kata kunci, dan z adalah jumlah total kemunculan semua pola yang ditemukan. Hal ini membuat Aho-Corasick jauh lebih unggul dibanding pencarian kata satu per satu, terutama ketika jumlah kata kunci sangat banyak. Dengan sekali pemrosesan teks, semua pola dapat ditemukan secara efisien tanpa perlu dilakukan pencarian berulang.

BAB 3

ANALISIS PEMECAHAN MASALAH

Bab ini akan menjelaskan tentang penguraian masalah dalam membangun sistem *Applicant Tracking System* berbasis CV digital. Pada analisis ini, akan dijelaskan langkah pemecahan masalah secara sistematis, pemetaan masalah ke algoritma yang akan digunakan, arsitektur aplikasi, dan ilustrasi penggunaannya.

3.1 Langkah Pemecahan Masalah

1. Mempersiapkan dataset (*seeding*)

Seeding dapat dilakukan dengan menggunakan kode yang telah disediakan oleh asisten. *Seeding* ini akan mensimulasikan kondisi ketika program telah memiliki sejumlah *applicant* sehingga simulasi dari algoritma dapat lebih terlihat dengan baik.

2. Ekstraksi teks dari file PDF

Setiap CV yang dikumpulkan akan berupa *file* PDF, sehingga program harus bisa mengubah tiap file ini menjadi sebuah *string* yang panjang. *String* yang dihasilkan ini akan digunakan pada algoritma-algoritma selanjutnya.

3. Implementasi algoritma *String Matching*

Bagian ini merupakan algoritma yang krusial untuk program ini. Pada awalnya, akan dilakukan teknik *exact matching*, yang mencari sebuah *substring* yang sama persis dengan *pattern* atau pola yang ingin dicari. Algoritma yang digunakan adalah algoritma KMP dan BM. Apabila program tidak dapat menemukan *substring* yang memenuhi kriteria, maka program akan beralih ke teknik *fuzzy matching*.

Mode *fuzzy matching* akan menggunakan algoritma *Levenshtein Distance*, dengan tujuan untuk menghitung tingkat kemiripan antara pola yang ingin dicari dengan teks CV. Tujuan dari *fuzzy matching* adalah untuk mengantisipasi terjadinya kesalahan ketik oleh pengguna (*typo*), sehingga program masih dapat mencoba untuk mencari teks yang cukup mendekati pola yang dicari oleh pengguna. Nilai ambang batas untuk menentukan *typo* ini juga ditentukan melalui pengujian agar nilainya optimal.

4. Implementasi *regex* untuk mencari informasi detail

Regex akan digunakan untuk menyaring informasi detail pada teks CV. Informasi yang didapatkan dengan menggunakan *regex* juga akan digunakan untuk membuat *summary* yang nantinya akan ditampilkan oleh program.

5. Penyampaian hasil dalam bentuk *website*

Hasil pencarian akan ditampilkan dalam bentuk kartu CV. Selain itu, pengguna juga dapat menekan tombol *summary* untuk mendapatkan rangkuman dari CV seseorang, atau bisa juga melihat CV aslinya dalam bentuk PDF.

3.2 Pemetaan masalah menjadi elemen-elemen algoritma *String Matching*

Dari sebuah database PDF, tidak bisa langsung digunakan algoritma *string matching* untuk mencari pola yang diinginkan. Maka dari itu, diperlukan pemetaan dari sebuah file PDF menjadi elemen-elemen yang dapat digunakan untuk menerapkan algoritma *string matching*.

1. Teks

Dari sebuah CV yang disimpan dalam bentuk file PDF, akan dibentuk sebuah string panjang yang akan berperan sebagai teks yang digunakan dalam algoritma *string matching*.

2. Pola (*Pattern*)

Pola yang digunakan akan didapatkan dari masukan pengguna, kemudian akan menjadi pola yang dicari pada sebuah CV yang telah diubah menjadi teks, dengan menggunakan algoritma *string matching*.

3. Pilihan algoritma yang ingin digunakan

Algoritma *string matching* yang akan digunakan, juga merupakan pilihan dari pengguna. Program akan menyediakan tiga metode *string matching*, yaitu Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick.

4. Hasil pencarian

Ada kemungkinan bahwa tidak ada teks CV yang memiliki *substring* yang sama persis seperti input dari pengguna. Maka dari itu, akan digunakan *Levenshtein Distance*, untuk menentukan teks termirip dengan pencarian *substring* tersebut. Dengan demikian, hasil yang dikembalikan adalah teks yang memiliki pola sama, atau teks yang paling mirip dengan pola yang dicari.

3.3 Fitur Fungsional dan Arsitektur Aplikasi

Aplikasi yang dibangun adalah sebuah Applicant Tracking System (ATS) berbasis desktop yang dikembangkan dengan bahasa pemrograman Python dan antarmuka grafis menggunakan pustaka Flet. Arsitektur sistem dirancang untuk mengelola dan menganalisis CV digital secara efisien. Secara garis besar, arsitektur ini terdiri dari tiga komponen utama: antarmuka pengguna (GUI), mesin pencarian, dan basis data.

3.3.1 Antarmuka Pengguna (GUI)

GUI menyediakan berbagai komponen wajib, termasuk kolom input untuk memasukkan beberapa kata kunci (dipisahkan koma), tombol untuk memilih algoritma pencarian exact match antara Knuth-Morris-Pratt, Boyer-Moore, atau Aho-Corasick, selektor untuk menentukan jumlah CV teratas yang ingin ditampilkan, serta tombol "Search" untuk memulai proses. Setelah pencarian selesai, hasil akan ditampilkan dalam bentuk kartu-kartu CV yang memuat nama kandidat, jumlah kata kunci yang cocok, dan frekuensinya. Setiap kartu dilengkapi tombol untuk melihat ringkasan (Summary) atau dokumen CV asli (View CV).

3.3.2 Mesin Pencarian

Ini adalah inti dari aplikasi yang menjalankan semua logika pencarian. Saat pengguna memulai pencarian, mesin akan melakukan langkah-langkah berikut:

1. Ekstraksi Teks dari PDF: Sistem akan mengambil setiap CV berformat PDF dari direktori data/ dan mengubah seluruh isinya menjadi satu string teks tunggal untuk dianalisis.
2. *Exact Matching*: Menggunakan algoritma Knuth-Morris-Pratt, Boyer-Moore, atau Aho-Corasick yang dipilih pengguna, sistem mencari kemunculan persis dari setiap kata kunci di dalam teks CV.
3. *Fuzzy Matching*: Jika ada kata kunci yang tidak ditemukan melalui *exact matching*, sistem secara otomatis akan melakukan pencarian lanjutan menggunakan algoritma Levenshtein Distance untuk menemukan kata-kata yang paling mirip, dengan mempertimbangkan kemungkinan kesalahan ketik (*typo*).
4. Ekstraksi Informasi: Untuk halaman ringkasan, sistem menggunakan *Regular Expression (Regex)* untuk mengekstrak informasi spesifik yaitu tanggal lahir, alamat, nomor telepon, keahlian, pengalaman kerja, dan riwayat pendidikan dari teks CV.

3.3.3 Basis Data dan Penyimpanan

Sistem menggunakan MySQL untuk menyimpan data terstruktur. Skema basis data terdiri dari tabel ApplicantProfile (menyimpan data diri pelamar) dan ApplicationDetail (menyimpan detail lamaran seperti posisi dan path ke file CV). Atribut cv_path pada tabel ApplicationDetail berfungsi sebagai penunjuk lokasi fisik file CV dalam format PDF yang tersimpan di dalam repositori sistem. Semua proses pencocokan dilakukan secara *in-memory* untuk memastikan kecepatan dan responsivitas.

3.4 Ilustrasi Penggunaan

Untuk memberikan gambaran yang lebih jelas mengenai alur kerja aplikasi, berikut adalah contoh kasus penggunaannya.

1. Seorang perekrut dari sebuah perusahaan teknologi ingin mencari kandidat untuk posisi "Data Scientist". Ia membutuhkan seorang kandidat yang menguasai "Python", memiliki pengalaman dengan "Machine Learning", dan familiar dengan "SQL".
2. Perekrut membuka aplikasi ATS, lalu memasukkan kata kunci Python, Machine Learning, SQL pada kolom input. Ia memilih algoritma Boyer-Moore dan menentukan untuk menampilkan Top 5 kandidat yang paling relevan. Setelah itu, ia menekan tombol Search.

3. Setelah tombol Search ditekan, sistem mulai memindai seluruh CV yang tersimpan di database. Setiap file PDF dikonversi menjadi teks. Algoritma yang bersangkutan dijalankan untuk mencari kata-kata kunci tersebut di setiap CV. Misalkan, dari hasil pemindaian *exact matching*, tidak ada CV yang mengandung kata kunci "Machine Learning" secara persis. Namun, sistem mendeteksi adanya frasa seperti "deep learning" atau salah ketik "machin lerning". Sistem kemudian mengaktifkan *fuzzy matching* menggunakan Levenshtein Distance untuk kata kunci "Machine Learning" dan menemukan kandidat yang memiliki frasa paling mirip.
4. Setelah proses pencarian selesai, Aplikasi akan menampilkan waktu eksekusi pencarian, misalnya: "Exact Match: 100 CVs scanned in 120ms. Fuzzy Match: 100 CVs scanned in 150ms." Lima kartu CV yang paling relevan akan ditampilkan, diurutkan berdasarkan jumlah total kecocokan kata kunci. Kartu teratas mungkin menampilkan:

Nama Kandidat: Aland Kecocokan: 3 kata kunci ditemukan Rincian: Python (10 kemunculan), SQL (8 kemunculan), Machine Learning (kemiripan ditemukan: 5) Tombol "Summary" dan "View CV".
--

5. Jika perekrut tertarik dengan profil Aland, perekrut dapat menekan tombol "Summary". Sebuah halaman baru akan muncul, menampilkan:

Informasi Pribadi: Nama, kontak, dan alamat (diambil dari basis data). Keahlian: <i>Python, SQL, TensorFlow, PyTorch, Scikit-learn</i> (diekstrak dengan <i>Regex</i>). Pengalaman Kerja: Data Scientist di Perusahaan XYZ (2022-2024) (diekstrak dengan <i>Regex</i>). Pendidikan: S1 Teknik Informatika, Institut Teknologi Bandung (diekstrak dengan <i>Regex</i>).
--

Dengan alur ini, perekrut dapat dengan cepat menyaring ratusan CV dan mengidentifikasi kandidat yang paling sesuai dengan kebutuhan tanpa harus memeriksa setiap dokumen secara manual.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Penjelasan teknis (data, kode/fungsi)

1. Knuth-Morris-Pratt

```
def knuth_morris_pratt(text: str, keywords: list[str]) ->
list[dict]:
    if not keywords:
        return []
    results = []
    for keyword in keywords:
        if not keyword:
            continue

        border = border_function(keyword)
        j = 0
        count = 0

        for i in range(len(text)):
            while j > 0 and text[i] != keyword[j]:
                j = border[j - 1]
            if text[i] == keyword[j]:
                j += 1
            if j == len(keyword):
                count += 1
                j = border[j - 1]

        if count == 0:
            continue
        results.append({"keyword": keyword, "occurrences":
count})

    return results
```

2. Boyer-Moore

```

def boyer_moore(text: str, keywords: list[str]) ->
list[dict]:
    if not keywords:
        return []

    results = []

    for keyword in keywords:
        if not keyword:
            continue

        last_occurrence = get_last_occ(keyword)
        i = 0
        count = 0

        while i <= len(text) - len(keyword):
            j = len(keyword) - 1

            while j >= 0 and text[i + j] == keyword[j]:
                j -= 1

            if j < 0:
                count += 1
                i += 1
            else:
                mismatch_char = text[i + j]
                last = last_occurrence.get(mismatch_char,
-1)

                i += max(1, j - last)

            if count == 0:
                continue

        results.append({"keyword": keyword,
"occurrences": count})

    return results

```

3. Aho-Corasick

```

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

class AhoCorasick:
    def __init__(self):
        self.root = AhoCorasickNode()

    def build_trie(self, keywords):
        """Build the trie structure from keywords"""
        for keyword in keywords:
            node = self.root
            for char in keyword.lower(): # Case
insensitive
                if char not in node.children:
                    node.children[char] =
AhoCorasickNode()
                node = node.children[char]
                node.is_end = True
                node.output.append(keyword)

    def build_failure_function(self):
        """Build failure function using BFS"""
        queue = deque()

        # Initialize failure function for depth 1 nodes
        for child in self.root.children.values():
            child.failure = self.root
            queue.append(child)

        # Build failure function for deeper nodes
        while queue:
            current = queue.popleft()

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

```

```

        queue.append(child)

        # Find failure node
        failure_node = current.failure
        while failure_node is not None and char
not in failure_node.children:
            failure_node = failure_node.failure

        if failure_node is not None:
            child.failure =
failure_node.children[char]
        else:
            child.failure = self.root

        # Add output from failure node
        child.output.extend(child.failure.output)

    def search(self, text):
        """Search for all occurrences of keywords in
text"""
        matches = defaultdict(int)
        node = self.root

        for i, char in enumerate(text.lower()): # Case
insensitive
            # Follow failure links until we find a match
or reach root
            while node is not None and char not in
node.children:
                node = node.failure

            if node is None:
                node = self.root
                continue

            node = node.children[char]

            # Add all matches ending at current position
            for keyword in node.output:

```

```

        matches[keyword] += 1

    return matches

def aho_corasick(text: str, keywords: list) -> list:
    if not keywords or not text:
        return []

    # Filter out empty keywords
    keywords = [k for k in keywords if k.strip()]
    if not keywords:
        return []

    # Build Aho-Corasick automaton
    ac = AhoCorasick()
    ac.build_trie(keywords)
    ac.build_failure_function()

    # Search for patterns
    matches = ac.search(text)

    # Format output
    keywords_data = []
    for keyword in keywords:
        occurrences = matches.get(keyword, 0)
        if occurrences != 0:
            keywords_data.append({
                "keyword": keyword,
                "occurrences": occurrences
            })

    return keywords_data

```

4. Regex

```

import fitz # PyMuPDF
import re

```

```

def parse_skills(skills_block):
    """Mem-parsing blok teks skills menjadi sebuah list
    yang lebih bersih."""
    if not skills_block:
        return []

    # Hapus judul bagian seperti 'Skills' atau
    'Highlights' dari blok itu sendiri
    skills_block = re.sub(r'(?i)skills|highlights', '',
skills_block)
    # Ganti baris baru dan bullet dengan koma
    cleaned_text = re.sub(r'[\n•*-', ', ', skills_block)

    # Pisahkan, bersihkan, dan hapus item yang tidak
    relevan/kosong
    skills_list = [skill.strip() for skill in
cleaned_text.split(',') if len(skill.strip()) > 1]
    return skills_list

def parse_education(education_block):
    """Mem-parsing blok teks education dengan logika yang
    lebih baik."""
    if not education_block:
        return []

    education_list = []
    # Pola untuk mengenali baris yang memulai entri
    pendidikan
    entry_pattern =
r'(?i)(bachelor|master|associate|phd|diploma|b\.s|m\.s|b\.
a)'

    current_entry_lines = []
    for line in education_block.strip().split('\n'):
        if re.search(entry_pattern, line) and
current_entry_lines:
            entry_text = ' '.join(current_entry_lines)
            edu_dict = {}

```

```

        year_match = re.search(r'(\d{4})', entry_text)
        if year_match:
            edu_dict['year'] = year_match.group(1)

            # Sisa teks adalah gelar dan institusi
            degree_text =
entry_text.replace(edu_dict.get('year', ''), '').strip()
            parts = degree_text.split(',')
            edu_dict['degree'] = parts[0].strip()
            edu_dict['institution'] =
            ','.join(parts[1:]).strip() if len(parts) > 1 else ''

            education_list.append(edu_dict)
            current_entry_lines = [line]
        else:
            current_entry_lines.append(line)

    # Proses entri terakhir yang tersisa
    if current_entry_lines:
        entry_text = ' '.join(current_entry_lines)
        edu_dict = {}
        year_match = re.search(r'(\d{4})', entry_text)
        if year_match:
            edu_dict['year'] = year_match.group(1)

            degree_text =
entry_text.replace(edu_dict.get('year', ''), '').strip()
            parts = degree_text.split(',')
            edu_dict['degree'] = parts[0].strip()
            edu_dict['institution'] =
            ','.join(parts[1:]).strip() if len(parts) > 1 else ''
            education_list.append(edu_dict)

    return education_list

def parse_experience(experience_block):
    """Mem-parsing blok teks experience dengan pemisah
    entri yang lebih cerdas."""
    if not experience_block:

```



```

        return []

        splitter_pattern =
r'\n(?:[A-Z][a-z\s]+.*\n(?:January|February|March|April|Ma
y|June|July|August|September|October|November|December|Jan
|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)\s+\d{4})'
        entries = re.split(splitter_pattern,
experience_block.strip())

        job_list = []
        year_pattern =
r'(?i)((?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
[a-z]*\s\d{4}\s*to\s*(?:Current|(?:Jan|Feb|Mar|Apr|May|Jun
|Jul|Aug|Sep|Oct|Nov|Dec)[a-z]*\s\d{4}))'

        for entry in entries:
            if not entry.strip() or len(entry.strip()) < 20: #
Abaikan entri yang terlalu pendek
                continue

            lines = [line.strip() for line in
entry.split('\n') if line.strip()]
            job_dict = {}

            job_dict['position'] = lines[0]

            year_match = re.search(year_pattern, entry)
            job_dict['year'] = year_match.group(0) if
year_match else ''

            # Gabungkan semua baris menjadi satu untuk
deskripsi
            full_desc_text = ' '.join(lines[1:])
            # Hapus info tahun dari deskripsi
            if job_dict['year']:
                full_desc_text =
full_desc_text.replace(job_dict['year'], '')
            job_dict['description'] = re.sub(r'\s+', ' ',
full_desc_text).strip()

```

```

        job_list.append(job_dict)

    return job_list

def process_cv(full_text):

    # Pola Regex untuk Ekstraksi Blok (termasuk
    # 'Highlights' untuk skills)
    pattern_experience = r"(?i)(?:Experience|Professional
Experience|Work
Experience)\b([\s\S]*?)(?=\n(?:Education|Skills|Highlights
|Projects|Qualifications|Accomplishments|Awards)\b|\Z)"
    pattern_education = r"(?i)(?:Education|Education and
Training)\b([\s\S]*?)(?=\n(?:Experience|Skills|Highlights|
Projects|Qualifications|Accomplishments|Awards)\b|\Z)"
    pattern_skills =
r"(?i)(?:Skills|Highlights)\b([\s\S]*?)(?=\n(?:Experience|
Education|Projects|Qualifications|Accomplishments|Awards)\
b|\Z)"

    # Ekstrak setiap blok
    experience_block_match = re.search(pattern_experience,
full_text)
    education_block_match = re.search(pattern_education,
full_text)
    skills_block_match = re.search(pattern_skills,
full_text)

    # Parsing setiap blok
    skills = parse_skills(skills_block_match.group(1) if
skills_block_match else None)
    education =
parse_education(education_block_match.group(1) if
education_block_match else None)
    job_history =
parse_experience(experience_block_match.group(1) if
experience_block_match else None)

```

```

cv_data = {
    "skills": skills,
    "education": education,
    "job_history": job_history
}

return cv_data

```

5. Seeder

```

import sys
sys.path.append("src")

from models.model import *
from core.database import create_tables
from core.service import *
from pathlib import Path
from faker import Faker
import random

fake = Faker()
fake.seed_instance(69) # Seed Faker for reproducibility

def seed_cv_to_db(path: Path, application_role: str):
    """
    Fungsi untuk memasukkan data sebuah CV ke dalam
    database.
    """
    applicants = get_all_applicants()
    chosen_applicant = random.choice(applicants)
    insert_application(
        applicant_id=chosen_applicant.applicant_id, #
type: ignore
        application_role=application_role,
        cv_path=path.absolute().as_posix()
    )

```

```

def seed_applicants(num_applicants: int):
    """
    Fungsi untuk membuat sejumlah data pelamar palsu dan
    memasukkannya ke dalam database.
    """
    global first_names, last_names, date_of_births,
    addresses, phone_numbers

    first_names = [fake.unique.first_name() for _ in
    range(num_applicants)]
    last_names = [fake.last_name() for _ in
    range(num_applicants)]
    date_of_births = [fake.date_of_birth(minimum_age=18,
    maximum_age=50) for _ in range(num_applicants)]
    addresses = [fake.address() for _ in
    range(num_applicants)]
    phone_numbers = [fake.unique.phone_number() for _ in
    range(num_applicants)]
    for i in range(num_applicants):
        insert_applicant(
            first_name=first_names[i],
            last_name=last_names[i],
            date_of_birth=date_of_births[i],
            address=addresses[i],
            phone_number=phone_numbers[i]
        )

def process_cv_files():
    """
    Melakukan iterasi pada folder data,
    mengambil 20 file PDF pertama secara leksikografis
    dari setiap kategori,
    dan memprosesnya.
    """

    # --- Mendapatkan direktori data ---
    data_dir = Path("data")
    if not data_dir.is_dir():

```

```

        print(f"Error: Direktori data utama '{data_dir}'
tidak ditemukan.")
        return

    # --- Iterasi pada setiap subdirektori dalam direktori
data ---
    try:
        categories = sorted([d for d in data_dir.iterdir()
if d.is_dir()])
        except Exception as e:
            print(f"Tidak dapat membaca direktori kategori:
{e}")
            return

        print(f"Menemukan {len(categories)} kategori di
direktori data. Memulai pemrosesan...")
        for category in categories:
            print(f"Memproses kategori: {category.name}")
            try:
                # --- Mengambil semua file PDF dalam kategori
---

                pdf_files = sorted(category.glob("*.pdf"))

                for pdf_file in pdf_files[:20]:
                    seed_cv_to_db(pdf_file, category.name)

                # hapus file pdf yang tidak dipilih
                for pdf_file in pdf_files[20:]:
                    try:
                        pdf_file.unlink() # Hapus file yang
tidak dipilih
                    except Exception as e:
                        print(f"Error saat menghapus file
'{pdf_file}': {e}")

                except Exception as e:
                    print(f"Error saat memproses kategori
'{category.name}': {e}")

```

```

if __name__ == "__main__":
    create_tables()
    seed_applicants(200)
    process_cv_files()
    print("Seeding completed successfully.")

```

6. Model

```

from sqlalchemy import Column, Integer, String, Date,
Text, ForeignKey
from sqlalchemy.orm import relationship
from core.database import Base

class ApplicantProfile(Base):
    __tablename__ = "applicantprofile"

    applicant_id = Column(Integer, primary_key=True,
index=True, nullable=False)
    first_name = Column(String(50), default=None,
nullable=True)
    last_name = Column(String(50), default=None,
nullable=True)
    date_of_birth = Column(Date, default=None,
nullable=True)
    address = Column(String(255), default=None,
nullable=True)
    phone_number = Column(String(50), default=None,
nullable=True)

    # --- Relationships ---
    applications = relationship("ApplicationDetail",
back_populates="applicant")

    # for debugging and logging purposes
    def __repr__(self):
        return f"<ApplicantProfile(id={self.applicant_id},
name='{self.first_name} {self.last_name}')"

```

```

class ApplicationDetail(Base):
    __tablename__ = "applicationdetail"

    detail_id = Column(Integer, primary_key=True,
index=True, nullable=False)
    applicant_id = Column(Integer,
ForeignKey("applicantprofile.applicant_id"),
nullable=False)
    application_role = Column(String(100), default=None,
nullable=True) # VARCHAR(100) DEFAULT NULL
    cv_path = Column(Text, default=None, nullable=True) #
TEXT DEFAULT NULL (TEXT can store long strings)

    # --- Relationships ---
    applicant = relationship("ApplicantProfile",
back_populates="applications")

    # for debugging and logging purposes
    def __repr__(self):
        return f"<ApplicationDetail(id={self.detail_id},
role='{self.application_role}',
applicant_id={self.applicant_id})>"

```

7. Database

```

from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base
# from sqlalchemy.ext.declarative import declarative_base
import os
from dotenv import load_dotenv

load_dotenv()

# --- Database Configuration ---
DATABASE_URL = os.getenv("DATABASE_URL",
"mysql+pymysql://root:password@localhost:3306/ats_database

```

```

")

# --- SQLAlchemy Engine Setup ---
engine = create_engine(
    DATABASE_URL, echo=False
)

# --- SQLAlchemy Session Setup ---
SessionLocal = sessionmaker(autocommit=False,
    autoflush=False, bind=engine)

# --- Base for Declarative Models ---
Base = declarative_base()

def get_db():
    """
    Provides a database session.
    Ensures the session is closed after use.
    """
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

# --- Function to create all tables ---
def create_tables():
    Base.metadata.drop_all(bind=engine)
    Base.metadata.create_all(bind=engine)
    print("Database tables created.")

```

8. Repository

```

from sqlalchemy.orm import Session
from models.model import ApplicantProfile,
    ApplicationDetail

# --- ApplicantProfile Repository Functions ---

```



```

def repo_get_all_applicants(db: Session):
    """
    Retrieve all applicants from the database.
    """
    return db.query(ApplicantProfile).all()

def repo_get_applicant_by_id(db: Session, applicant_id:
int):
    """
    Retrieve an applicant by their ID.
    """
    return
db.query(ApplicantProfile).filter(ApplicantProfile.applica
nt_id == applicant_id).first()

def repo_insert_applicant(db: Session, applicant:
ApplicantProfile):
    """
    Insert a new applicant into the database.
    """
    db.add(applicant)
    db.commit()
    db.refresh(applicant)
    return applicant

# kayanya gaboleh update/delete ga sih

# --- ApplicationDetail Repository Functions ---

def repo_get_all_applications(db: Session):
    """
    Retrieve all applications from the database.
    """
    return db.query(ApplicationDetail).all()

def repo_get_applications_by_applicant_id(db: Session,
applicant_id: int):
    """

```

```

        Retrieve all applications for a specific applicant.
        """
        return
db.query(ApplicationDetail).filter(ApplicationDetail.applicant_id == applicant_id).all()

def repo_insert_application(db: Session, application: ApplicationDetail):
    """
    Insert a new application into the database.
    """
    db.add(application)
    db.commit()
    db.refresh(application)
    return application

```

9. Service

```

from core.database import get_db
from core.repository import *
from datetime import date
from core.algorithm import knuth_morris_pratt, boyer_moore, aho_corasick, fuzzy_match
from core.utils import extract_text_from_pdf
import time
import pickle
import json
import os
from pathlib import Path
from core.regex import process_cv

cv_data_text = {}
CACHE_FILE = Path("data/cache/cv_data_cache.pkl")

def save_cache():
    """Save cv_data_text to cache file"""
    CACHE_FILE.parent.mkdir(parents=True, exist_ok=True)
    with open(CACHE_FILE, 'wb') as f:

```

```

        pickle.dump(cv_data_text, f)

def load_cache():
    """Load cv_data_text from cache file"""
    global cv_data_text
    try:
        if CACHE_FILE.exists():
            with open(CACHE_FILE, 'rb') as f:
                cv_data_text = pickle.load(f)
            return True
        except (FileNotFoundError, pickle.PickleError,
EOFError):
            cv_data_text = {}
            return False

def extract_all_cv_data(force_refresh=False):
    """
    Extract all CV data with caching support
    """
    global cv_data_text

    # Try to load from cache first (unless force refresh)
    if not force_refresh and load_cache():
        print(f"Loaded {len(cv_data_text)} CV records from
cache")
        return cv_data_text

    print("Extracting CV data from PDFs...")
    from core.service import get_all_applications

    cv_data_text = {} # Reset the dictionary
    applications = get_all_applications()

    for application in applications:
        if application.cv_path and
application.applicant_id: # type: ignore
            cv_text =
extract_text_from_pdf(application.cv_path)
            cv_text = cv_text.lower()

```

```

        if cv_text:
            cv_data_text[application.detail_id] = {
                "applicant_id":
application.applicant_id,
                "cv_text": cv_text,
                "cleaned_text": cv_text.replace('\n',
' ').strip()
            }

    # Save to cache
    save_cache()
    print(f"Extracted and cached {len(cv_data_text)} CV
records")
    return cv_data_text

def clear_cache():
    """Clear the cache file"""
    if CACHE_FILE.exists():
        CACHE_FILE.unlink()
        print("Cache cleared")

# --- Service Functions for ApplicantProfile ---

def get_all_applicants():
    """
    Retrieve all applicants from the database.
    """
    db = next(get_db())
    try:
        return repo_get_all_applicants(db)
    finally:
        db.close()

def get_applicant_by_id(applicant_id: int):
    """
    Retrieve an applicant by their ID.
    """
    db = next(get_db())
    try:

```

```

        return repo_get_applicant_by_id(db, applicant_id)
    finally:
        db.close()

def insert_applicant(first_name: str, last_name: str,
date_of_birth: date, address: str, phone_number: str):
    """
    Insert a new applicant into the database.
    """
    db = next(get_db())
    try:
        applicant = ApplicantProfile(
            first_name=first_name,
            last_name=last_name,
            date_of_birth=date_of_birth,
            address=address,
            phone_number=phone_number
        )
        return repo_insert_applicant(db, applicant)
    finally:
        db.close()

# --- Service Functions for ApplicationDetail ---

def get_all_applications():
    """
    Retrieve all applications from the database.
    """
    db = next(get_db())
    try:
        return repo_get_all_applications(db)
    finally:
        db.close()

def get_applications_by_applicant_id(applicant_id: int):
    """
    Retrieve all applications for a specific applicant.
    """

```

```

        db = next(get_db())
        try:
            return repo_get_applications_by_applicant_id(db,
applicant_id)
        finally:
            db.close()

def insert_application(applicant_id: int,
application_role: str, cv_path: str):
    """
    Insert a new application into the database.
    """
    db = next(get_db())

    try:
        applicant = repo_get_applicant_by_id(db,
applicant_id)
        if not applicant:
            return None

        application = ApplicationDetail(
            applicant_id=applicant_id,
            application_role=application_role,
            cv_path=cv_path
        )
        return repo_insert_application(db, application)
    finally:
        db.close()

def get_cv_data_by_applicant_id(applicant_id: int):
    """
    Retrieve the CV data for a specific applicant.
    """
    db = next(get_db())
    try:
        applications =
repo_get_applications_by_applicant_id(db, applicant_id)
        if not applications:

```

```

        return None
        applicant = repo_get_applicant_by_id(db,
applicant_id)
        if not applicant:
            return None

        regex_res =
process_cv(cv_data_text.get(applications[0].detail_id,
{})).get("cv_text", "")

        cv_data = {
            "name": f"{applicant.first_name}
{applicant.last_name}",
            "birthdate":
applicant.date_of_birth.isoformat() if
applicant.date_of_birth else None, # type: ignore
            "address": applicant.address,
            "phone_number": applicant.phone_number,
            "skills": regex_res.get("skills", []),
            "education": regex_res.get("education", []),
            "job_history": regex_res.get("job_history",
[]),
        }

        return cv_data

    finally:
        db.close()

def search_matching_data(keywords: list[str], algo: str,
top_match: int) -> dict:
    applicants = get_all_applicants()

    applicant_match_count = 0
    applicants_results = []
    chosen_applications = set()

    curr_time = time.time()

```

```

    for applicant in applicants:
        if (applicant_match_count >= top_match) and
(top_match > 0):
            break
        applications =
get_applications_by_applicant_id(applicant.applicant_id) #
type: ignore
        for application in applications:
            results = []
            cv_text =
cv_data_text.get(application.detail_id,
{}).get("cleaned_text", "")
            if not cv_text:
                continue
            if algo == "Knuth-Morris-Pratt":
                results = knuth_morris_pratt(cv_text,
keywords)
            elif algo == "Boyer-Moore":
                results = boyer_moore(cv_text, keywords)
            elif algo == "Aho-Corasick":
                results = aho_corasick(cv_text, keywords)
            else:
                raise ValueError(f"Unknown algorithm:
{algo}")

            if not results:
                continue

            applicant_match_count += 1
            applicants_results.append({
                "applicant_id": applicant.applicant_id,
                "name": f"{applicant.first_name}
{applicant.last_name}",
                "matched_keywords": len(results),
                "keywords_data": results,
                "cv_path": application.cv_path,
                "bgcolor": "#E3F2FD" # Example background
color

```



```

    ))
    chosen_applications.add(application.detail_id)

    if (applicant_match_count >= top_match) and
(top_match > 0):
        break

    exact_match_stats = {
        "count": applicant_match_count,
        "time_ms": int((time.time() - curr_time) * 1000)
# Convert to milliseconds
    }

    # Fuzzy matching
    fuzzy_match_count = 0
    curr_time = time.time()
    for applicant in applicants:
        if (applicant_match_count >= top_match) and
(top_match > 0):
            break
        applications =
get_applications_by_applicant_id(applicant.applicant_id) #
type: ignore
        for application in applications:
            cv_text =
cv_data_text.get(application.detail_id,
{}).get("cleaned_text", "")
            if not cv_text:
                continue

            if not application.detail_id or
application.detail_id in chosen_applications: # type:
ignore
                continue

            results = fuzzy_match(cv_text, keywords)

            if not results:
                continue

```

```

        fuzzy_match_count += 1
        applicants_results.append({
            "applicant_id": applicant.applicant_id,
            "name": f"{applicant.first_name}
{applicant.last_name}",
            "matched_keywords": len(results),
            "keywords_data": results,
            "cv_path": application.cv_path,
            "bgcolor": "#E3F2FD" # Example background
color
        })

        if (applicant_match_count >= top_match) and
(top_match > 0):
            break

    fuzzy_match_stats = {
        "count": fuzzy_match_count,
        "time_ms": int((time.time() - curr_time) * 1000)
# Convert to milliseconds
    }

    results_data = {
        "exact_match_stats": exact_match_stats,
        "fuzzy_match_stats": fuzzy_match_stats,
        "applicants": applicants_results
    }

    return results_data

# results_data = {
#     "exact_match_stats": {"count": 0, "time_ms": 0},
#     "fuzzy_match_stats": {"count": 0, "time_ms": 0},
#     "applicants": []
# }

```

10. Utils

4.2 Penggunaan

CV Analyzer App

Analyze and search through CVs efficiently

Search Configuration

Keywords:

Search Algorithm:

Knuth-Morris-Pratt

Boyer-Moore

Aho-Corasick

Top Matches:

Search

Gambar 4.2.1. Halaman utama

Search Configuration

Keywords:

Search Algorithm:

Knuth-Morris-Pratt

Boyer-Moore

Aho-Corasick


Top Matches:

Search

Results

Exact Match: 0 CVs scanned in 318ms.

Fuzzy Match: 0 CVs scanned in 2918ms.



No Applicants Found

We couldn't find any applicants matching your search criteria.

Gambar 4.2.2. Tidak ditemukan aplikasi

Search Configuration

Keywords:

Search Algorithm:

Knuth-Morris-Pratt

Boyer-Moore

Aho-Corasick

Top Matches:

Search

Results

Exact Match: 10 CVs scanned in 16ms.

Fuzzy Match: 0 CVs scanned in 0ms.

Moh4mm4d Nu9r4h4

1 matched keyword

1. rea: 4 occurrences

Summary

View CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. rea: 3 occurrences

Summary

View CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. rea: 11 occurrences

Summary

View CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. rea: 5 occurrences

Summary

View CV

Tugas Besar 3 IF2211 Strategi Algoritma – Semester II Tahun 2024/2025

43

Gambar 4.2.3. Ditemukan aplikasi

[← Back to Search](#)[CV Summary](#)

Moh4mm4d Nu9r4h4

Birthdate: 2003-06-14

Address: Jl. Kenanga No. 12, Jakarta

Phone: N/A

Skills

and creative support with team

building and the ability work independently. i adapt to

change quickly and motivate staff to ensure a smooth workflow and increased productivity.

typing 55 wpm/ten

key 10500 kspm

microsoft office (word

excel

Education

diploma : data entry specialists eldorado college - city

state s.e.l.f high school - city , state

1995

Job History

No job history data available

[View Full CV](#)

Gambar 4.2.4. Summary

16407619.pdf1 / 2100%+

CUSTOMER SERVICE REPRESENTATIVE
Professional Summary
Office professional with an extensive background of more than twelve years of Customer Service, Account Manager and Office Administrative Support. Exude strong and excellent communications skills and creative support with team-building and the ability work independently. I adapt to change quickly and motivate staff to ensure a smooth workflow and increased productivity.

Skills

- Typing 55 wpm/Ten-Key 10500 kspm
- Microsoft Office (Word, Excel, Outlook)
- Data Entry/Customer Liaison/Support
- Agency Management System (AMS, 360, Saginata)
- The Agency Manager (TAM) Applied System
- CRM Software Systems
- Quick learner

Work History
Customer Service Representative 09/2017 to Current
Company Name â€ City , State

- Provide customer support to accounts, track orders, provided price quotes, order changes and/or cancellations.
- Identify customers' needs, research issue and complaints with problem solving for resolution.
- Assist to ensure professional and exceptional customer service with products inquiry and online services.
- Document account and conversation during inbound and outbound calls in call center environment.

Owner 01/2015 to 09/2017
Company Name â€ City , State

- Responsible for day-to-day operations of online retail store, including sales, stock and resource management.
- Develop social media strategy and set goals to increase brand awareness and engagement.
- Maintained online storefront and social media platforms with new products and marketing sales promotions.

Commercial/Personal Lines Account Manager 03/2014 to 01/2015
Company Name â€ City , State

- Temporary assignment ended March 214 - Jan.
- A Processed Certificates of Insurance for heavy contractor's, service and retail risk for commercial Line policies
- Verified and explained Commercial Insurance policy coverages and issued renewals
- Processed endorsements, cancellations, and reinstatements of policies and file claims with carrier
- Followed-Up on policy change request issued by carriers and/or sub agents.

Insurance Customer Associate 02/2011 to 10/2013
Company Name â€ City , State

- Office location closed Feb.011- Oct.
- 2013 Provided customer service for retail brokerage firm as an inbound call center representative to new and existing policyholders and

Gambar 4.2.1. Melihat PDF full

4.3 Pengujian

CV Analyzer App

Analyze and search through CVs efficiently

Search Configuration

Keywords:

Search Algorithm:

Knuth-Morris-Pratt

Boyer-Moore

Aho-Corasick

Top Matches:

Search

Results

Exact Match: 0 CVs scanned in 324ms.

Fuzzy Match: 0 CVs scanned in 8619ms.

Gambar 4.3.1. TestCase

Search Configuration

Keywords:

education

Search Algorithm:

Knuth-Morris-Pratt

Boyer-Moore

Aho-Corasick

Top Matches:

5

Search

Results

Exact Match: 5 CVs scanned in 2ms.

Fuzzy Match: 0 CVs scanned in 0ms.

Moh4mm4d Nu9r4h4

1 matched keyword

1. education: 1 occurrence

SummaryView CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. education: 1 occurrence

SummaryView CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. education: 1 occurrence

SummaryView CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. education: 1 occurrence

SummaryView CV

Gambar 4.3.2. TestCase

Search Configuration

Keywords:

work

Search Algorithm:

Knuth-Morris-Pratt

Boyer-Moore

Aho-Corasick

Top Matches:

20

Search

Results

Exact Match: 20 CVs scanned in 19ms.

Fuzzy Match: 0 CVs scanned in 0ms.

Moh4mm4d Nu9r4h4

1 matched keyword

1. work: 7 occurrences

SummaryView CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. work: 11 occurrences

SummaryView CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. work: 32 occurrences

SummaryView CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. work: 2 occurrences

SummaryView CV

Gambar 4.3.3. TestCase

Search Configuration

Keywords:

experience

Search Algorithm:

Knuth-Morris-Pratt

Boyer-Moore

Aho-Corasick

Top Matches:

5

Search

Results

Exact Match: 5 CVs scanned in 6ms.

Fuzzy Match: 0 CVs scanned in 0ms.

Moh4mm4d Nu9r4h4

1 matched keyword

1. experience: 1 occurrence

SummaryView CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. experience: 3 occurrences

SummaryView CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. experience: 1 occurrence

SummaryView CV

Moh4mm4d Nu9r4h4

1 matched keyword

1. experience: 1 occurrence

SummaryView CV

Gambar 4.3.4. TestCase

4.4 Analisis hasil pengujian

Berdasarkan hasil testing, dapat dilihat bahwa apabila tidak ditemukan *string* yang sama, maka akan membutuhkan waktu pencarian yang lebih. Selain itu, pencarian yang dilakukan dengan menggunakan pola yang panjang, juga akan membutuhkan waktu yang lebih lama karena kompleksitas waktu saat melakukan comparing.

BAB 5

KESIMPULAN, SARAN, DAN REFLEKSI

5.1 Kesimpulan

Screening CV secara manual membutuhkan waktu yang cukup lama dan rentan terhadap subjektivitas manusia, terutama ketika dihadapkan pada ribuan lamaran. Maka dari itu, dibuat sebuah sistem *Applicant Tracking System* (ATS) berbasis CV digital yang memanfaatkan teknologi *pattern matching* untuk mempersingkat waktu pencarian kandidat berdasarkan kata kunci. Aplikasi yang dibangun mampu mengubah dokumen CV dalam format PDF menjadi string teks panjang untuk dianalisis. Secara keseluruhan, aplikasi ini berhasil memenuhi tujuan utamanya untuk mempermudah dan mempercepat proses rekrutmen dengan menyediakan alat penyaringan kandidat yang efektif dan berbasis data.

5.2 Saran

Dalam membangun sistem pencarian seperti ini, sangat disarankan untuk tidak melakukan proses yang sama berulang kali. Proses konversi dari PDF ke teks untuk setiap CV sebaiknya tidak dijalankan setiap kali pengguna menekan tombol cari. Hasil konversi teks tersebut sebaiknya disimpan (*in-memory*) agar bisa digunakan kembali pada pencarian-pencarian berikutnya, sehingga bisa meningkatkan efisiensi dan responsivitas aplikasi secara keseluruhan.

Selain itu, untuk meningkatkan akurasi ekstraksi informasi, disarankan untuk melakukan pengembangan lebih lanjut dengan memanfaatkan *Natural Language Processing* (NLP). Penggunaan *Regex* saat ini sangat bergantung pada pola teks yang tetap untuk mengekstrak informasi seperti keahlian atau pengalaman kerja. Hal ini bisa menjadi kurang efektif jika format CV sangat beragam. Dengan NLP, sistem dapat dilatih untuk "memahami" konteks kalimat sehingga mampu mengenali berbagai informasi penting secara lebih cerdas dan fleksibel, terlepas dari tata letak CV.

5.3 Refleksi

Dari pengerjaan, dapat ditarik kesimpulan bahwa kerja hari-h tidak efektif. ***Terbukti.***

LAMPIRAN

Tautan Repository Github

https://github.com/albertchriss/Tubes3_micinkriuk

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

DAFTAR PUSTAKA

CP-Algorithms. (2023). Aho-Corasick algorithm. Diakses pada 15 Juni 2025, dari https://cp-algorithms.com/string/aho_corasick.html

CP-Algorithms. (2025). Prefix function. Diakses pada 15 Juni 2025, dari <https://cp-algorithms.com/string/prefix-function.html>