

LAPORAN TUGAS BESAR 3

IF2211 STRATEGI ALGORITMA

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



Disusun Oleh:

Rafif Farras 13523095

Barru Adi Utomo 13523101

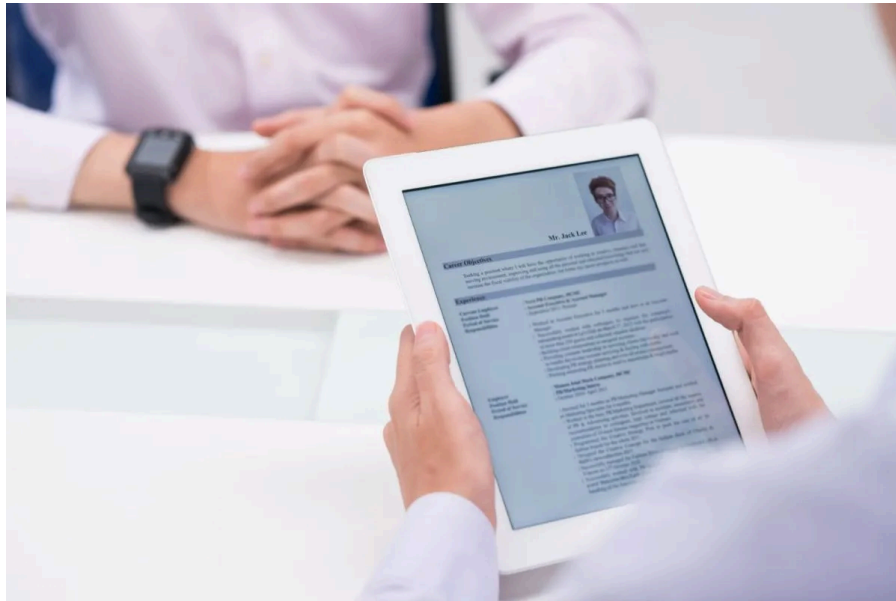
Azfa Radhiyya Hakim 13523115

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

DAFTAR ISI

I. Deskripsi Tugas.....	3
II. Landasan Teori.....	5
2.1. Algoritma Knuth-Morris-Pratt.....	5
2.2. Algoritma Boyer-Moore.....	5
2.3. Algoritma Aho-Corasick.....	6
2.4. Levenshtein Distance.....	6
2.5. Regular Expression (Regex).....	6
2.6. Enkripsi.....	6
2.7. Graphical User Interface.....	7
III. Analisis Pemecahan Masalah.....	8
3.1. Extract Text.....	8
3.2. Algoritma Pattern Matching.....	8
3.2.1 Knuth-Morris-Pratt.....	8
3.2.2 Boyer-Moore.....	9
3.2.3 Aho-Corasick.....	9
3.2.4 Levenshtein Distance.....	10
3.3. Enkripsi.....	11
3.4. Regex.....	12
3.5. Fitur Fungsional GUI.....	13
3.6. Contoh Ilustrasi Kasus.....	14
IV. Implementasi dan Testing.....	16
4.1. Struktur File.....	16
4.2. Class dan Metode.....	17
4.3. Tata Cara Penggunaan.....	27
4.4. Analisis Hasil Pengujian.....	31
V. PENUTUP.....	36
5.1. Kesimpulan.....	36
5.2. Saran.....	36
5.3. Refleksi.....	36
LAMPIRAN.....	38
Tautan Repository Github.....	38
Tautan Video.....	38
Tabel Kelengkapan Spesifikasi.....	38
DAFTAR PUSTAKA.....	39

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.

II. Landasan Teori

2.1. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) adalah sebuah algoritma pencocokan string yang efisien untuk menemukan semua kemunculan sebuah pola (*pattern*) di dalam sebuah teks. Keunggulan utama KMP terletak pada kemampuannya untuk menghindari perbandingan karakter yang berulang. Algoritma ini lebih efisien dibandingkan dengan algoritma *brute force*, dan akan mencapai efisiensinya dengan melakukan prapemrosesan pada *pattern* untuk membuat sebuah tabel bantu yang disebut tabel *Longest Proper Prefix* (LPS) atau fungsi pinggiran yang menyimpan panjang dari prefiks (awalan) terpanjang dari *pattern* yang juga merupakan sufiks (akhiran) untuk setiap substring *pattern*.

Saat terjadi ketidakcocokan antara karakter di teks dan *pattern*, algoritma KMP tidak selalu menggeser *pattern* satu per satu. Sebaliknya, ia menggunakan nilai dari tabel LPS untuk menentukan seberapa jauh pergeseran *pattern* dapat dilakukan tanpa melewatkan kemungkinan adanya kecocokan. Dengan cara ini, KMP melompati bagian-bagian teks yang sudah pasti tidak akan cocok, sehingga mengurangi jumlah total perbandingan secara signifikan, terutama pada kasus di mana *pattern* memiliki sub-pola yang berulang.

2.2. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah algoritma pencocokan string yang lebih cepat dari KMP. Berbeda dengan KMP yang membandingkan dari kiri ke kanan, Boyer-Moore melakukan perbandingan dari karakter paling kanan pada *pattern*. Kecepatan algoritma ini didasarkan pada dua heuristik utama:

1. ***Bad Character Heuristic***: Saat terjadi ketidakcocokan, algoritma melihat karakter pada teks yang menyebabkan ketidakcocokan tersebut (*bad character*). Kemudian, *pattern* digeser ke kanan hingga karakter *bad character* menjadi benar. Jika *bad character* tersebut tidak ada sama sekali di dalam *pattern*, maka *pattern* dapat digeser sejauh panjangnya sendiri.
2. ***Good Suffix Heuristic***: Heuristik ini digunakan saat terjadi ketidakcocokan setelah beberapa karakter di bagian kanan *pattern* sudah cocok. Bagian yang sudah cocok ini disebut *good suffix*. Algoritma akan menggeser *pattern* untuk mencari kemunculan lain dari *good suffix* tersebut di dalam *pattern* itu sendiri.

Dengan menggabungkan kedua heuristik ini dan mengambil nilai pergeseran yang paling maksimal, Boyer-Moore sering kali dapat melakukan lompatan yang sangat besar di sepanjang teks, menjadikannya salah satu algoritma pencocokan string yang paling efisien.

2.3. Algoritma Aho–Corasick

Algoritma Aho-Corasick adalah ekstensi dari KMP yang dirancang untuk mencari semua kemunculan dari beberapa pattern sekaligus di dalam sebuah teks dalam satu kali proses.

Algoritma ini bekerja dengan membangun struktur data Trie dari semua *pattern* yang ingin dicari. Setelah Trie terbentuk, algoritma menambahkan *failure links* (mirip dengan tabel LPS pada KMP) untuk setiap node. *Failure link* dari sebuah node akan menunjuk ke node lain di dalam Trie yang merepresentasikan prefiks terpanjang dari string yang diwakili oleh node tersebut, yang juga merupakan sufiks dari string lain di dalam Trie. Saat melakukan pencocokan pada teks, jika terjadi ketidakcocokan karakter, algoritma akan mengikuti *failure link* untuk berpindah ke *state* lain tanpa perlu memulai ulang dari awal teks. Ini memungkinkan Aho-Corasick untuk menemukan semua pattern secara paralel dalam waktu yang sebanding dengan panjang teks.

2.4. Levenshtein Distance

Levenshtein Distance adalah sebuah metrik yang digunakan untuk mengukur perbedaan antara dua urutan string. Jarak ini didefinisikan sebagai jumlah minimum operasi penyuntingan satu karakter yang diperlukan untuk mengubah satu kata menjadi kata lainnya. Terdapat tiga jenis operasi penyuntingan yang diizinkan:

- Penyisipan (*Insertion*): Menambahkan satu karakter ke dalam string.
- Penghapusan (*Deletion*): Menghapus satu karakter dari string.
- Penggantian (*Substitution*): Mengganti satu karakter dengan karakter lainnya.

Algoritma ini umumnya diimplementasikan menggunakan pemrograman dinamis dengan membangun sebuah matriks berukuran $(m+1) \times (n+1)$, di mana m dan n adalah panjang dari kedua string. Setiap sel dalam matriks merepresentasikan jarak Levenshtein antara *substring* dari kedua string. Nilai pada sel terakhir matriks adalah jarak Levenshtein total antara kedua string.

2.5. Regular Expression (Regex)

Regular Expression adalah sebuah sekuens karakter yang mendefinisikan sebuah pola pencarian. Pola ini digunakan untuk mencocokkan, menemukan, mengelola, dan manipulasi teks. Regex menyediakan sintaks yang sangat kuat dan fleksibel untuk mendeskripsikan pola teks yang kompleks, jauh melampaui kemampuan pencocokan string biasa.

2.6. Enkripsi

Enkripsi adalah proses mengubah data dari format yang dapat dibaca (plaintext) menjadi format yang tidak dapat dibaca dan terkode (ciphertext). Proses ini menggunakan sebuah algoritma enkripsi dan sebuah kunci (key) untuk mengamankan data. Tujuan utama enkripsi adalah untuk

memastikan kerahasiaan (*confidentiality*), di mana data hanya dapat diakses oleh pihak yang memiliki kunci dekripsi yang sesuai.

2.7. Graphical User Interface

GUI (*Graphical User Interface*) adalah antarmuka pengguna berbasis grafis agar pengguna dapat berinteraksi dengan perangkat lunak melalui elemen visual seperti tombol, ikon, menu, dan teks. GUI dirancang agar lebih intuitif dan mudah digunakan, bahkan oleh pengguna non-teknis tidak seperti antarmuka berbasis teks (*command line interface*), .

III. Analisis Pemecahan Masalah

3.1. Extract Text

Algoritma ini bertujuan untuk mengonversi konten dari sebuah dokumen PDF menjadi format teks mentah (*string*) yang ternormalisasi. Proses ini penting sebagai tahap pra-pemrosesan sebelum teks dapat dianalisis lebih lanjut. Secara garis besar, pengubahan ini dilakukan dengan cara seperti berikut.

1. Mengubah seluruh teks menjadi huruf kecil (*lowercase*).
2. Menghapus semua karakter non-alfanumerik (selain huruf dan angka), kecuali spasi.
3. Mengganti karakter baris baru (*newline*) dengan spasi.
4. Membersihkan spasi putih yang berlebih.

Contoh pdf yang sudah di normalisasi

accountant summary financial accountant specializing in financial planning reporting and analysis within the department of defense highlights account reconciliations results oriented financial reporting critical thinking accounting operations professional analysis of financial systems erp enterprise resource planning software excellent facilitator accomplishments served
--

3.2. Algoritma Pattern Matching

3.2.1 *Knuth-Morris-Pratt*

Algoritma yang dibuat didasarkan pada flow sebagai berikut.

- Membuat fungsi LSP yang berukuran sama dengan panjang pola dan diinisialisasi dengan 0
- Inisialisasi variabel $length = 0$ (untuk melacak panjang prefix-suffix yang cocok sebelumnya) dan $i = 1$ (sebagai pointer untuk iterasi pada pola).
- Proses pembuatan tabel LSP, yaitu lakukan perulangan pada i selama masih kecil daripada panjang pola
- Jika karakter $pola[i]$ sama dengan $pola[length]$, tambah nilai $length$ dengan 1 dan simpan nilai $length$ baru ke dalam $lsp[i]$
- Jika karakter pada tahap sebelumnya tidak sama, jika $length$ tidak sama dengan 0, artinya ada prefix-suffix yang cocok sebelumnya. Proses dilakukan dengan "mundur" dengan mengubah $length$ menjadi nilai yang ada di $lsp[length - 1]$. Namun jika $length$ sama dengan 0, set $lsp[i]$ menjadi 0 dan lanjutkan ke karakter berikutnya dengan menambah i dengan 1

- Proses pencarian, inisialisasi variabel j sebagai index pointer dari pola dimulai dari 0. Bandingkan nilai $pola[j]$ dengan $teks[i]$. Jika sama, maka masing-masing dari i dan j ditambah dengan 1
- Jika berbeda, jika j bukan sama dengan 0, perbarui j dengan nilai $lsp[j-1]$. Jika j sama dengan 0, nilai dari i ditambah dengan 1
- Proses pencarian dilakukan ketika nilai j sama dengan panjang dari pattern. Jika sudah mencapai akhir dari teks namun belum ditemukan kecocokan, menyimpulkan tidak ada pattern yang sesuai di dalam teks.

3.2.2 *Boyer-Moore*

Algoritma yang dibuat didasarkan pada flow sebagai berikut.

- Pertama, dibuat fungsi praproses “Bad Character” dengan membuat tabel yang mencatat posisi kemunculan terakhir setiap karakter pada pattern
- Inisialisasi posisi awal pencarian pada teks dengan $i = 0$
- Lakukan pengulangan tahap-tahap berikut selama nilai i masih memungkinkan pola untuk dimuat pada sisa teks
- Pada setiap tahapannya, pencocokan dimulai dari karakter pattern terakhir (paling kanan) dengan menginisialisasi variabel $pattern_position$ sama dengan panjang dari pattern dikurang 1. Secara iterasi, perbandingan dilakukan secara mundur ke karakter awal pada pattern dengan mengurangi $pattern_position$ dengan 1
- Jika nilai dari i kurang dari 0, menandakan pencocokan berhasil.
- Jika terjadi ketidakcocokan, hitung nilai pergeseran berdasarkan aturan “Bad Character”.
- Hitung nilai pergeseran berdasarkan aturan Good Suffix (mencari kemunculan lain dari bagian yang sudah cocok).
- Pilih nilai pergeseran yang paling besar di antara kedua aturan tersebut.
- Perbarui i dengan menambahkannya sesuai nilai pergeseran terbesar yang dipilih.

3.2.3 *Aho-Corasick*

Algoritma yang dibuat didasarkan pada flow sebagai berikut.

- Buat sebuah struktur data Trie dengan satu root node.
- Masukkan setiap pola dari list ke dalam Trie. Ini dilakukan dengan menelusuri Trie karakter per karakter dari setiap pola, dan membuat node baru jika jalur karakter belum ada.
- Tandai node terakhir dari setiap pola yang dimasukkan sebagai akhir dari sebuah kata dan simpan indeks dari pola tersebut.
- Buat Failure Link untuk setiap node di dalam Trie, yang berfungsi sebagai jalan pintas saat terjadi ketidakcocokan karakter.
- Inisialisasi $failure_link$ dari root ke dirinya sendiri, dan $failure_link$ dari semua node di level 1 (anak langsung dari root) ke root.

- Gunakan antrian (queue) untuk memproses semua node secara BFS untuk membangun failure_link dan output_link kepada seluruh Trie.
- Untuk setiap node, failure_link akan menunjuk ke node lain di Trie yang mewakili suffix terpanjang dari path node tersebut yang juga merupakan prefix dari suatu pola lain.
- Jika karakter dari teks tidak memiliki transisi dari current_node, ikuti failure_link secara berulang hingga ditemukan transisi yang cocok atau kembali ke root.
- Jika ada transisi yang cocok, pindahkan current_node ke node berikutnya.
- Setelah berpindah ke node baru, periksa apakah node tersebut menandai akhir sebuah pola.
- Tinjau rantai output_link dari current_node untuk menemukan semua pola yang berakhir pada posisi tersebut.

3.2.4 *Levenshtein Distance*

Algoritma yang dibuat didasarkan pada flow sebagai berikut.

- Buat sebuah matriks (tabel 2D) dengan ukuran (panjang string1 + 1) x (panjang string2 + 1) dan inisialisasi baris dan kolom pertama matriks dengan nilai urutan (0, 1, 2, ...), yang merepresentasikan biaya untuk mengubah string kosong menjadi string target.
- Lakukan perulangan untuk mengisi sisa sel matriks dari kiri atas ke kanan bawah.
- Jika karakter dari kedua string pada posisi yang dibandingkan sama, isi sel matriks dengan nilai dari sel diagonal kiri atas (tidak ada biaya).
- Jika karakter berbeda, isi sel dengan nilai 1 + minimum dari tiga sel tetangga: sel kiri (operasi penyisipan), sel atas (operasi penghapusan), dan sel diagonal (operasi penggantian).
- Nilai akhir di pojok kanan bawah matriks adalah hasil akhir Levenshtein Distance, yaitu jumlah minimum operasi untuk mengubah satu string menjadi string lainnya.
- Buat fungsi similarity untuk mengubah nilai distance menjadi skor kesamaan (0.0 hingga 1.0) dengan rumus $1 - (\text{distance} / \text{panjang string maksimal})$.
- Kemudian, membuat (sliding window berukuran panjang pattern di kali dengan 1.2
- Lakukan perulangan untuk setiap ukuran jendela, lalu geser jendela tersebut di sepanjang teks dari awal hingga akhir.
- Pada setiap posisi jendela, ambil potongan teks (substring) dan hitung similarity-nya terhadap pola menggunakan metode Levenshtein di atas.
- Jika skor similarity lebih tinggi dari ambang batas (threshold) yang ditentukan, simpan informasi kecocokan tersebut (posisi, panjang, dan skor).
- Threshold yang digunakan adalah 75% dengan alasan nilai ini adalah nilai yang sangat cocok ketika ada typo dalam memasukkan keyword. Misalkan user ingin mencari “cook”, dan tanpa sengaja menuliskan “coik”. Kasus seperti ini terhandle untuk nilai threshold tersebut.
- Setelah semua kemungkinan jendela diperiksa, urutkan semua kecocokan yang valid berdasarkan skor similarity dari yang tertinggi ke terendah.

3.3. Enkripsi

Algoritma yang dibuat didasarkan pada flow sebagai berikut.

1. Encryption

Algoritma ini dibuat untuk mengenkripsi teks asli dengan menambahkan hash berantai dan signature

- Inisialisasi prev_hash dengan nilai 0.
- Lakukan perulangan untuk setiap karakter dalam teks asli.
- Di dalam perulangan, buat sebuah hash unik untuk karakter saat ini yang juga bergantung pada prev_hash. Ini dibuat supaya terdapat efek berantai di mana enkripsi setiap karakter dipengaruhi oleh karakter sebelumnya.
- Buat sebuah key secara random.
- Hitung nilai ASCII karakter terenkripsi dengan rumus: $(\text{ASCII asli} + \text{ASCII kunci} + \text{hash saat ini}) \% 95 + 32$. Rumus ini memastikan hasilnya selalu dalam rentang karakter yang bisa dicetak (printable characters).
- Simpan karakter terenkripsi yang baru, lalu perbarui prev_hash dengan nilai hash saat ini untuk digunakan di iterasi berikutnya.
- Setelah semua karakter dienkripsi, hitung total dari semua hash yang dibuat, lalu buat signature 3 digit dari hasil tersebut sebagai penanda integritas data.
- Gabungkan semua karakter terenkripsi dan tambahkan signature 3 digit di bagian akhir sebagai hasil akhir.
- Ubah teks menjadi dalam bentuk base64

2. Decryption

- Pertama, decode terlebih dahulu teks dengan base64
- Pisahkan teks terenkripsi menjadi dua bagian: data utama dan signature 3 digit di akhir.
- Inisialisasi prev_hash dengan nilai 0.
- Lakukan perulangan untuk setiap karakter pada data utama yang terenkripsi.
- Proses dilakukan dengan mencoba semua kemungkinan karakter asli (dari ASCII 32 hingga 126). Untuk setiap percobaan, lakukan kembali proses enkripsi (membuat hash, mengambil karakter kunci, dan menghitung dengan rumus yang sama).
- Jika hasil enkripsi dari karakter percobaan cocok dengan karakter terenkripsi yang sedang diproses, maka karakter percobaan tersebut adalah karakter asli yang benar.
- Simpan karakter asli yang ditemukan, perbarui prev_hash dengan hash dari karakter yang benar, dan lanjutkan ke karakter berikutnya. Jika tidak ada yang cocok, proses dekripsi gagal.
- Setelah semua karakter berhasil didekripsi, hitung total dari semua hash yang direkonstruksi.

- Bandingkan total hash hasil dekripsi dengan nilai signature yang didapat di awal. Jika keduanya sama, maka data valid dan teks asli dikembalikan. Jika tidak, artinya data rusak atau kunci salah, dan proses gagal.

3.4. Regex

Disini, kami akan menjelaskan 3 fungsi utama yang dilakukan oleh regex, yaitu untuk mengekstrak summary, skills, experience, dan education.

1. Extract Summary

- Buat sebuah daftar stop_keywords yang berisi kata kunci untuk menandai akhir dari sebuah bagian ringkasan (contoh: "experience", "education", "skills").
- Gabungkan semua stop_keywords menjadi satu pola pencarian untuk digunakan nanti.
- Buat sebuah pola regex utama yang mencari header seperti "summary" atau "profile" (tidak peka huruf besar/kecil).
- Setelah menemukan header, regex akan menangkap semua teks (.+?)
- Proses penangkapan teks akan berhenti jika regex menemukan salah satu dari dua kondisi menggunakan lookahead (?=...):
- Jika kecocokan ditemukan, ambil teks ringkasan dan mengembalikan hasilnya

2. Extract Skills

- Buat dua pola regex: satu untuk mendeteksi header bagian skills (header_pattern), dan satu lagi untuk mendeteksi header yang menandai akhir bagian skills (end_section_pattern).
- Inisialisasi sebuah variabel penanda kondisi, misalnya in_skills_section = False.
- Lakukan perulangan untuk setiap baris dalam teks CV.
- Jika sebuah baris cocok dengan header_pattern, ubah kondisi menjadi in_skills_section = True
- Jika kondisi sedang True dan sebuah baris cocok dengan end_section_pattern, ubah kondisi kembali menjadi False untuk berhenti merekam.
- Selama kondisi in_skills_section adalah True, proses setiap baris yang ditemukan:
- Normalisasi baris dengan mengganti berbagai pemisah (seperti /, ;, •) menjadi koma.
- Pisahkan baris menjadi daftar keahlian individu berdasarkan koma.
- Terakhir, ubah set yang berisi semua keahlian menjadi sebuah daftar (list)

3. Extract Experience

- Definisikan sebuah pola regex untuk tanggal (`date_pattern`) yang dapat mengenali format umum seperti "Bulan Tahun to Bulan Tahun" atau "Bulan Tahun to Present".
- Definisikan juga pola `stop_headers` yang berisi kata kunci untuk menandai akhir dari sebuah deskripsi pekerjaan seperti `education`, `skills`, dll.
- Jika sebuah baris cocok dengan `date_pattern`, anggap ini adalah awal dari sebuah entri pengalaman baru.
- Ekstrak informasi dari baris tersebut, seperti jabatan dan rentang tanggal.
- Asumsikan baris berikutnya berisi nama perusahaan dan lokasi, lalu coba ekstrak informasi tersebut.
- Perulangan deskripsi ini akan berhenti jika menemukan baris kosong, `date_pattern` baru (entri pengalaman selanjutnya), atau `stop_headers`.
- Satukan semua informasi yang terkumpul (jabatan, perusahaan, tanggal, deskripsi) ke dalam sebuah dictionary dan tambahkan ke daftar pengalaman.

4. Extract Experience

- Menggunakan sebuah pola regex untuk menemukan dan menangkap seluruh blok teks di bawah header "Education".
- Blok ini akan ditangkap hingga regex menemukan baris kosong atau header bagian utama lainnya (contoh: "certifications", "skills").
- Jika blok "Education" tidak ditemukan, kembalikan daftar kosong.
- Jika ditemukan, pecah blok teks tersebut menjadi baris-baris individual.
- Definisikan sebuah `degree_keywords` yang berisi kata kunci gelar seperti "MBA", "Bachelors", "Ph.D".
- Jika sebuah baris mengandung salah satu dari `degree_keywords`, anggap baris ini sebagai satu entri riwayat pendidikan.
- Dari baris tunggal tersebut, dicoba untuk ekstrak informasi yang lebih detail seperti gelar, jurusan, institusi, dan lokasi dengan menggunakan pemisahan string dan pencarian regex yang lebih kecil.

Perlu diperhatikan bahwa metode regex yang digunakan **tidak sepenuhnya sempurna**. Hal ini disebabkan kondisi dari dokumen yang berbeda-beda dan tidak stabil.

3.5. Fitur Fungsional GUI

Teknologi yang digunakan untuk mengembangkan GUI adalah PyQt5. PyQt5 adalah framework Python untuk membangun antarmuka pengguna grafis (GUI) berbasis pustaka Qt.

Pada penerapan dalam kode, pengembangan PyQt dibagi menjadi beberapa bagian, yaitu:

- Components

Folder components berisi komponen yang dapat digunakan berulang kali dalam kode, seperti result_card.py, result_view.py, radio.py, sidebar.py, dan switch.py.

- Pages

Folder pages berisi file yang digunakan untuk halaman yang ada. Pages merupakan wrapper dari komponen-komponen yang ada. Pages yang tersedia adalah about_page, analyze_page, detail_page, landing_page, dan pdf_view.

- Assets

Folder assets berisi aset-aset yang digunakan dalam program, seperti foto anggota dan font yang digunakan.

- Main Program

Main program menyatukan page-page yang ada dan mengatasi perpindahan page, dengan *passing* fungsi change_page.

3.6. Contoh Ilustrasi Kasus

Berikut adalah contoh kasus penggunaan algoritma KMP, Boyer-Moore. Misalkan ingin dicari pattern “akuak” pada teks “kamusukaakuakatauyam”

1. Penggunaan KMP

Pertama, buat terlebih dahulu tabel LSP sebagai berikut.

a	k	u	a	k
0	0	0	1	2

Kemudian, pencocokan dapat dilakukan dengan cara berikut

k	a	m	u	s	u	k	a	a	k	u	a	k	a	t	a	u	a	y	a	m
a	k	u	a	k																
	a	k	u	a	k															
		a	k	u	a	k														
			a	k	u	a	k													
				a	k	u	a	k												

					a	k	u	a	k										
						a	k	u	a	k									
							a	k	u	a	k								
								a	k	u	a	k							

2. Penggunaan Boyer-Moore

Pertama, buat terlebih dahulu tabel Last Occurrence sebagai berikut.

a	k	u
3	4	2

Kemudian, pencocokan dapat dilakukan dengan cara berikut

k	a	m	u	s	u	k	a	a	k	u	a	k	a	t	a	u	a	y	a	m
a	k	u	a	k																
					a	k	u	a	k											
							a	k	u	a	k									
								a	k	u	a	k								

IV. Implementasi dan Testing

4.1. Struktur File

Berikut adalah struktur folder dan file yang kami gunakan.

```
.src
├── algorithms
│   ├── AhoCorasick.py
│   ├── BoyerMoore.py
│   ├── KnuthMorris.py
│   ├── Levenshtein.py
│   ├── ProfileEncryption.py
│   └── Regex.py
├── controller
│   └── Controller.py
├── database
│   ├── db_config.py
│   ├── db_init.py
│   ├── db_insert.py
│   ├── db_search.py
│   └── models.py
├── gui
│   ├── assets
│   │   └── Poppins-Regular.ttf
│   ├── components
│   │   ├── radio.py
│   │   ├── result_card.py
│   │   ├── result_view.py
│   │   ├── sidebar.py
│   │   └── switch.py
│   ├── main.py
│   ├── pages
│   │   ├── about_page.py
│   │   ├── analyze_page.py
│   │   ├── detail_page.py
│   │   ├── landing_page.py
│   │   └── pdf_view.py
```



```

|   |— Poppins
|   |   └─ Poppins-Regular.ttf
|— main.py
|— requirements.txt
└─ utils
    └─ normalize_pdf.py

```

4.2. Class dan Metode

Kami membuat beberapa metode dalam class yang digunakan sebagai pondasi utama jalannya aplikasi ini. Disini kami hanya menampilkan beberapa metode utama yang digunakan untuk setiap class.

1. Class KnuthMorris

Berisi implementasi dari algoritma Knuth Morris

Method kmp_algorithm

```

def kmp_algorithm(self, pattern: str):
    if not self.text or not pattern:
        return []

    lsp = self.identify_border_table(pattern)
    result_index = []

    i = 0
    j = 0

    while i < len(self.text):
        if pattern[j] == self.text[i]:
            i += 1
            j += 1

        if j == len(pattern):
            result_index.append(i - j)
            j = lsp[j - 1]
        elif i < len(self.text) and pattern[j] != self.text[i]:
            if j != 0:
                j = lsp[j - 1]
            else:
                i += 1

    return result_index

```

2. Class BoyerMoore

Berisi implementasi dari algoritma BoyerMoore

Metode boyer_moore_algorithm

```
def boyer_moore_algorithm(self, pattern: str):
    if not self.text or not pattern:
        return []

    found_positions = []
    text_length = len(self.text)
    pattern_length = len(pattern)
    bad_char_table = self.preprocess_bad_char(pattern)

    if pattern_length > text_length:
        return found_positions

    text_position = 0
    while text_position <= text_length - pattern_length:
        pattern_position = pattern_length - 1

        while pattern_position >= 0 and pattern[pattern_position] ==
self.text[text_position + pattern_position]:
            pattern_position -= 1

        if pattern_position < 0:
            found_positions.append(text_position)
            text_position += 1
        else:
            bad_char_shift_amt = self.bad_char_shift(bad_char_table,
self.text[text_position + pattern_position], pattern_position)
            good_suffix_shift_amt = self.good_suffix_shift(pattern, pattern_position)
            text_position += max(bad_char_shift_amt, good_suffix_shift_amt)

    return found_positions
```

3. Class Levenshtein

Berisi implementasi dari algoritma Levenshtein

Metode fuzzy_compare

```
# str1 text yang menjadi pembanding (pattern)
# str2 text panjang yang akan dibandingkan
# return adalah list index dan panjang kata yang persentasenya di atas threshold
@staticmethod
def fuzzy_compare(str1, str2, threshold):

    if not str1 or not str2:
        return []

    pattern = str1.lower().strip()
    text = str2.lower().strip()

    matches = []

    if len(pattern) >= len(text):
        max_len = max(len(pattern), len(text))
        distance = Levenshtein.distance(pattern, text)
```

```

        similarity = 1 - (distance / max_len)
        if similarity >= threshold:
            matches.append({
                'start': 0,
                'length': len(text),
                'similarity': similarity
            })
        return matches

pattern_len = len(pattern)

for i in range(len(text) - pattern_len + 1):
    substring = text[i:i + pattern_len]

    if substring == pattern:
        continue

    distance = Levenshtein.distance(pattern, substring)
    similarity = 1 - (distance / pattern_len)

    if similarity >= threshold:
        matches.append({
            'start': i,
            'length': pattern_len,
            'similarity': similarity
        })

extended_window = int(pattern_len * 1.2)
if extended_window <= len(text):
    for i in range(len(text) - extended_window + 1):
        substring = text[i:i + extended_window]

        if pattern in substring or substring in pattern:
            continue

        distance = Levenshtein.distance(pattern, substring)
        similarity = 1 - (distance / max(len(pattern), len(substring)))

        if similarity >= threshold:
            is_duplicate = False
            for j, match in enumerate(matches):
                if (i < match['start'] + match['length'] and i + extended_window >
match['start']):
                    if similarity > match['similarity']:
                        matches[j] = {
                            'start': i,
                            'length': extended_window,
                            'similarity': similarity
                        }
                    is_duplicate = True
                    break

            if not is_duplicate:
                matches.append({
                    'start': i,
                    'length': extended_window,
                    'similarity': similarity
                })

matches.sort(key=lambda x: x['similarity'], reverse=True)
return matches

```

4. Class AhoCoraisk (Bonus)

Berisi implementasi dari algoritma Knuth Morris

Methode AhoCoraisk

```
def search(self, text: str):
    text = text.lower()
    results = {pattern: 0 for pattern in self.patterns}
    current_node = self.root

    for i, char in enumerate(text):
        while char not in current_node.children and current_node is not self.root:
            current_node = current_node.failure_link

        if char in current_node.children:
            current_node = current_node.children[char]
        else:
            continue

        temp_node = current_node
        while temp_node is not None:
            if temp_node.is_end_of_word:
                pattern = self.patterns[temp_node.pattern_index]

                results[pattern] += 1

            temp_node = temp_node.output_link

    return results
```

5. Class Regex

Berisi implementasi dari algoritma regex yang digunakan untuk ekstraksi summary.

Methode extract_skills

```
def extract_skills(self):
    all_skills = set()
    header_pattern = r"(?i)^\s*(skills|technical\s+skills|core\s+competencies)\s*$"
    end_section_pattern = r"^\s*(professional
experience|education|affiliations|interests|languages|additional information|certification)"

    in_skills_section = False
    for line in self.text.split('\n'):
        line = line.strip()

        if re.match(header_pattern, line):
            in_skills_section = True
            continue

        if re.match(end_section_pattern, line, re.IGNORECASE) or not line:
            in_skills_section = False
            continue
```

```

    if in_skills_section:
        normalized_line = re.sub(r'\s*[;/;\.]\s*', ' ', line)

        normalized_line = re.sub(r'\s*\.(.*?\)', ' ', normalized_line)

        skills_list = normalized_line.split(',')

        for skill in skills_list:
            skill = skill.strip()
            if len(skill) > 2 and not skill.replace('.', '', 1).isdigit():
                all_skills.add(skill)

    return sorted(list(all_skills))

```

Method extract_experience

```

def extract_experience(self):
    experiences = []
    lines = self.text.splitlines()

    stop_headers = r"(?i)^(education|languages|skills|affiliations|references)"

    i = 0
    while i < len(lines):
        line = lines[i].strip()

        date_pattern =
r"(?i)((?:January|February|March|April|May|June|July|August|September|October|November|December|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)\s+\d{4}\s+to\s+(?:Current|Present|(?:January|February|March|April|May|June|July|August|September|October|November|December|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)\s+\d{4}))"
        date_match = re.search(date_pattern, line)

        if date_match:
            date_str = date_match.group(1)
            position = line.replace(date_str, "").strip().rstrip(',')

            try:
                start_date, end_date = [d.strip() for d in date_str.split(" to ")]
            except ValueError:
                start_date, end_date = date_str, "N/A"

            company_name = ""
            location = ""
            if i + 1 < len(lines):
                company_line = lines[i+1].strip()
                parts = company_line.split('-', 1)
                if len(parts) == 1:
                    parts = company_line.split('-', 1)

                if len(parts) == 2:
                    company_name = parts[0].strip()
                    location = parts[1].strip()
                else:
                    company_name = company_line

            description_lines = []
            j = i + 2

```

```

        while j < len(lines):
            desc_line = lines[j].strip()

            if not desc_line or re.search(date_pattern, desc_line) or
re.match(stop_headers, desc_line):
                break

            description_lines.append(desc_line.lstrip('• ').strip())
            j += 1

        experiences.append({
            "position": position,
            "company": company_name,
            "location": location,
            "start_date": start_date,
            "end_date": end_date,
            "description": description_lines
        })

        i = j - 1

    i += 1

    return experiences

```

Method extract_education

```

def extract_education(self):
    educations = []

    edu_section_pattern =
r"(?i)education\s*\n((?:.|\\n)+?) (?:\n\s*\n|Z|affiliations|certifications|skills)"
    edu_match = re.search(edu_section_pattern, self.text)

    if not edu_match:
        return []

    edu_text = edu_match.group(1).strip()
    lines = edu_text.split("\n")

    degree_keywords = r"(?i) (MBA|Bachelors\s*Degree|Master|Associate|Ph\.?D) "

    for line in lines:
        line = line.strip()

        if re.search(degree_keywords, line):
            entry = {
                "degree": "N/A",
                "field": "N/A",
                "institution": "N/A",
                "year": "N/A",
                "location": "N/A"
            }

            location_parts = line.split('-', 1)
            main_part = location_parts[0].strip()
            if len(location_parts) > 1:
                entry["location"] = location_parts[1].strip()

```

```

degree_match = re.search(degree_keywords, main_part)
if degree_match:
    degree_text = degree_match.group(0)
    entry["degree"] = degree_text.replace(',', ' ').strip()

    field_inst_text = main_part.replace(degree_text, "").strip()

    text_parts = field_inst_text.rsplit(' ', 2)
    if len(text_parts) > 2 and "college" in text_parts[1].lower():
        entry["institution"] = f"{text_parts[1]} {text_parts[2]}"
        entry["field"] = text_parts[0].replace(',', ' ').strip()
    else:
        entry["field"] = field_inst_text

    educations.append(entry)

return educations

```

6. Class ProfileEncryption (Bonus)

Berisi implementasi dari algoritma enkripsi dan dekripsi yang dibangun

Method _make_hash

```

def _make_hash(self, text: str):
    hash_num = 0
    for i, char in enumerate(text):
        hash_num += ord(char) * (i + 1)
    return hash_num % 1000

```

Method _make_block

```

def _make_block(self, char: str, prev_hash: int) -> dict:
    current_hash = self._make_hash(char + str(prev_hash))
    return {'data': char, 'prev_hash': prev_hash, 'hash': current_hash}

```

Method encrypt

```

def encrypt(self, text: str) -> str:
    if not text:
        return ""

    self.blocks = []
    encrypted_chars = []
    prev_hash = 0

    for i, char in enumerate(text):
        block = self._make_block(char, prev_hash)
        self.blocks.append(block)
        key_char = self.key[i % len(self.key)]
        encrypted_ascii = (ord(char) + ord(key_char) + block['hash']) % 95 + 32
        encrypted_chars.append(chr(encrypted_ascii))

```

```

        prev_hash = block['hash']

        total_hash = sum(block['hash'] for block in self.blocks) % 1000
        signature = f"{total_hash:03d}"

        return ''.join(encrypted_chars) + signature

```

Metode decrypt

```

def decrypt(self, encrypted_text: str):
    if len(encrypted_text) < 3:
        return ""

    signature = encrypted_text[-3:]
    data = encrypted_text[:-3]

    try:
        expected_total = int(signature)
    except:
        return ""

    self.blocks = []
    decrypted_chars = []
    prev_hash = 0

    for i, encrypted_char in enumerate(data):

        for test_ascii in range(32, 127):
            test_char = chr(test_ascii)

            test_block = self._make_block(test_char, prev_hash)

            key_char = self.key[i % len(self.key)]
            expected_encrypted = ord(test_char) + ord(key_char) + test_block['hash']
            expected_encrypted = expected_encrypted % 95 + 32

            if chr(expected_encrypted) == encrypted_char:

                self.blocks.append(test_block)
                decrypted_chars.append(test_char)
                prev_hash = test_block['hash']
                break

        else:
            return ""

    actual_total = sum(block['hash'] for block in self.blocks) % 1000

    if actual_total == expected_total:
        return ''.join(decrypted_chars)
    else:
        return ""

```


7. Class Controller

Berfungsi menghubungkan GUI dengan algoritma pattern matching

Method searchQuery

```
def searchQuery(self, pattern: str, algorithm: str, max: int):
    results = {}
    for i, data in enumerate(self.allData):
        path = data.get('cv_path', '')
        document_text = self.mapPathAndData.get(path, '')
        if algorithm == 'kmp':
            self.kmp.set_text(document_text)
            foundedList = self.kmp.find_multiple_keywords_kmp(pattern.lower())
        elif algorithm == 'booye':
            self.booye.set_text(document_text)
            foundedList = self.booye.find_multiple_keywords_bm(pattern.lower())
        elif algorithm == 'aho':
            self.aho.set_pattern(pattern.lower())
            foundedList = self.aho.search(document_text)

        if foundedList:
            results[i] = {
                "applicant_id": data.get('applicant_id', ''),
                "cv_path": path,
                "name": f"{data.get('first_name', '')} {data.get('last_name', '')}",
                "count": sum(foundedList.values()),
                "keywords_count": foundedList
            }

    if not results:
        for i, data in enumerate(self.allData):
            path = data.get('cv_path', '')
            document_text = self.mapPathAndData.get(path, '')
            self.kmp.set_text(document_text)
            foundedList = Levenshtein.find_multiple_keywords_fuzzy(pattern.lower(),
document_text)

            if foundedList:
                results[i] = {
                    "applicant_id": data.get('applicant_id', ''),
                    "cv_path": path,
                    "name": f"{data.get('first_name', '')} {data.get('last_name',
'' )}",
                    "keywords_count": foundedList,
                    "count": len(foundedList)
                }

        sorted_list = sorted(results.items(), key=lambda item: item[1]['count'],
reverse=True)

        return dict(sorted_list[:max])
```

8. Class db_search

Berisi implementasi untuk mengambil data dari database

Methodode getAllData

```
def getAllData(self):
    conn = self.get_connection()
    cursor = conn.cursor(dictionary=True)
    query = """
    SELECT ap.*, ad.*
    FROM ApplicantProfile ap
    JOIN ApplicationDetail ad ON ap.applicant_id = ad.applicant_id
    """
    cursor.execute(query)
    results = cursor.fetchall()
    cursor.close()
    return results
```

9. Class MainWindow

Logika utama dalam menampilkan GUI

Class MainWindow

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("CV Analyzer ATS")
        self.resize(1000, 700)

        outer_container = QWidget()
        outer_container.setStyleSheet("background-color: #FFF8DC;")
        outer_layout = QVBoxLayout(outer_container)
        outer_layout.setContentsMargins(80, 80, 80, 80)

        central_widget = QWidget()
        central_widget.setStyleSheet("background-color: #F4F4F4; border-radius: 15px;")
        central_layout = QVBoxLayout(central_widget)

        self.stack = QStackedWidget()
        self.page_landing = LandingPage(self.change_page)
        self.page_analyze = AnalyzePage(self.change_page)
        self.page_detail = DetailPage(self.change_page)

        self.stack.addWidget(self.page_landing) # index 0
        self.stack.addWidget(self.page_analyze) # index 1
        self.stack.addWidget(self.page_detail) # index 2

        central_layout.addWidget(self.stack)
        outer_layout.addWidget(central_widget)
```

```

self.setCentralWidget(outer_container)

def change_page(self, index, path=None, id=None):
    if index == 2 and path is not None:
        self.page_detail.load_path(path, id)
    self.stack.setCurrentIndex(index)

```

4.3. Tata Cara Penggunaan

1. Lakukan clone repository yang terdapat pada [lampiran](#).
2. Unduh dataset CV yang ingin digunakan. Dapat menggunakan [link](#) berikut. Pastikan folder “data” berada di root proyek
3. Ini adalah proses setup. Jika belum memiliki database, repository ini menyediakan seeding yang dapat membuat tabel secara otomatis. Pertama, buat sebuah database bernama “cv_ats” pada MySQL. Masuk ke file src/algorithm/ProfileEncryption, dan sesuaikan nama database, user, host, dan password yang digunakan. Jika sudah sesuai, jalankan seeding dengan memasukkan command berikut.

Command seeding atribut database dengan enkripsi

```
> python3 "src/algorithms/ProfileEncryption.py"
```

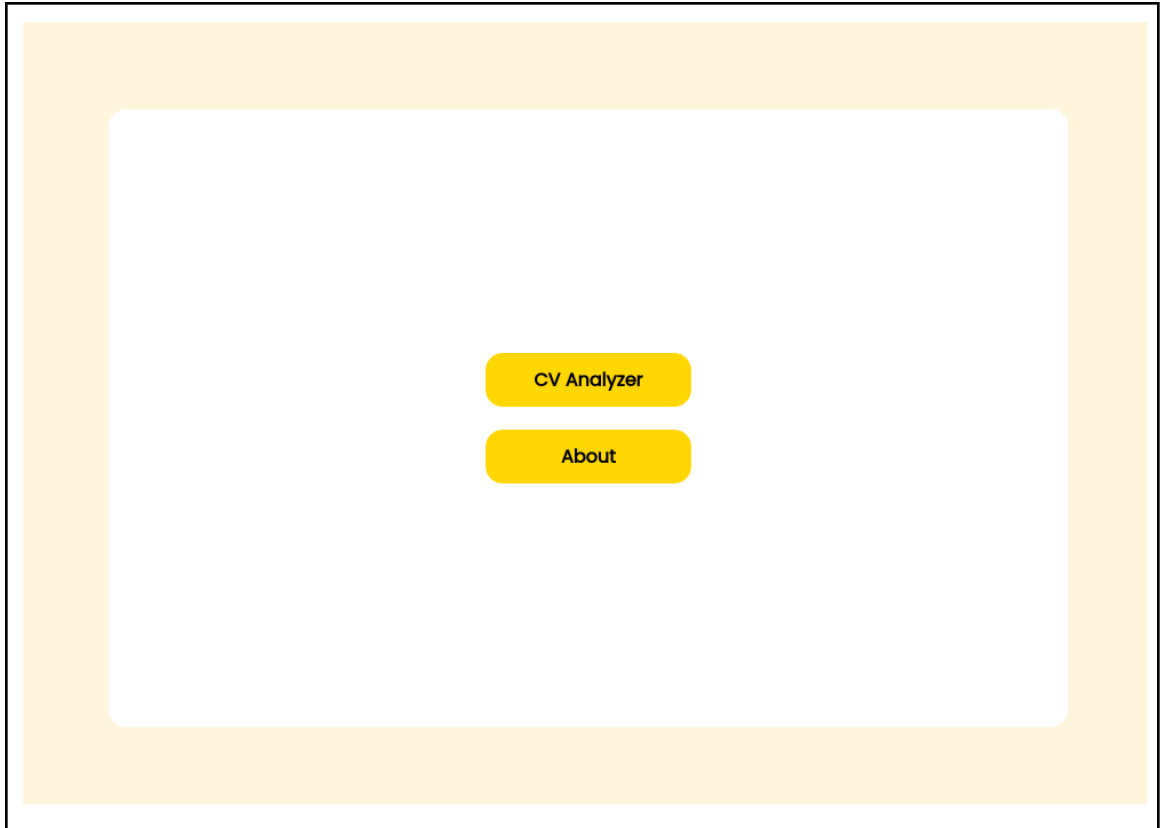
4. Jalankan aplikasi dengan command berikut untuk menampilkan interface

Command menjalankan aplikasi

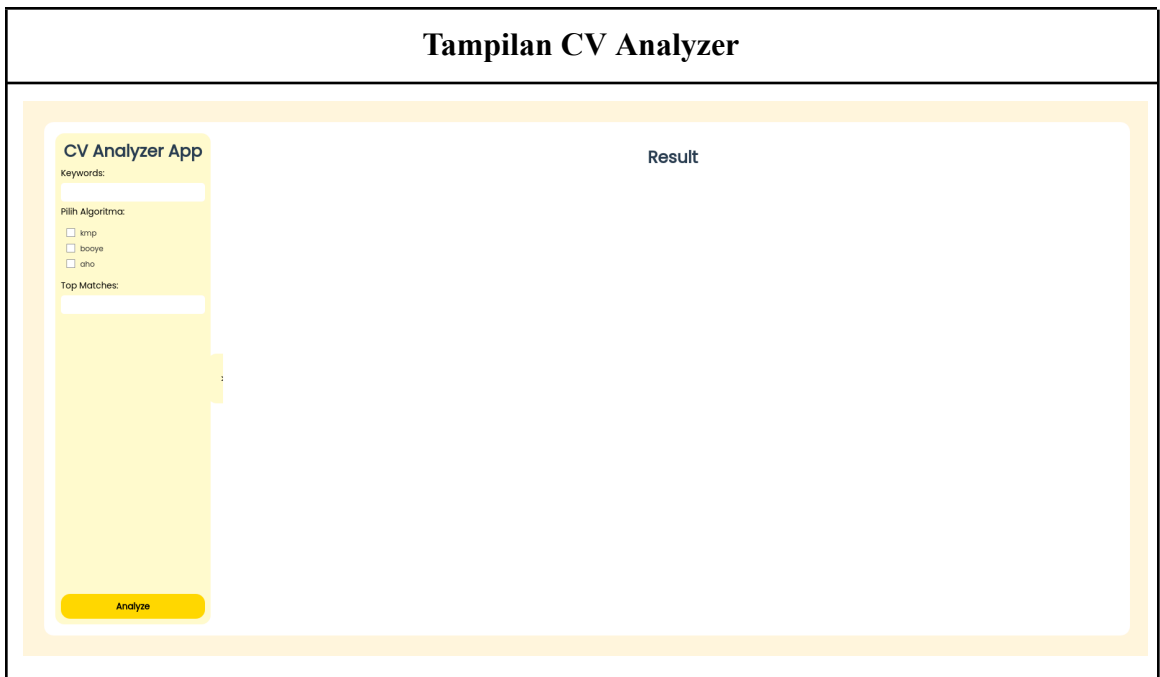
```
> python3 -m src.gui.main
```

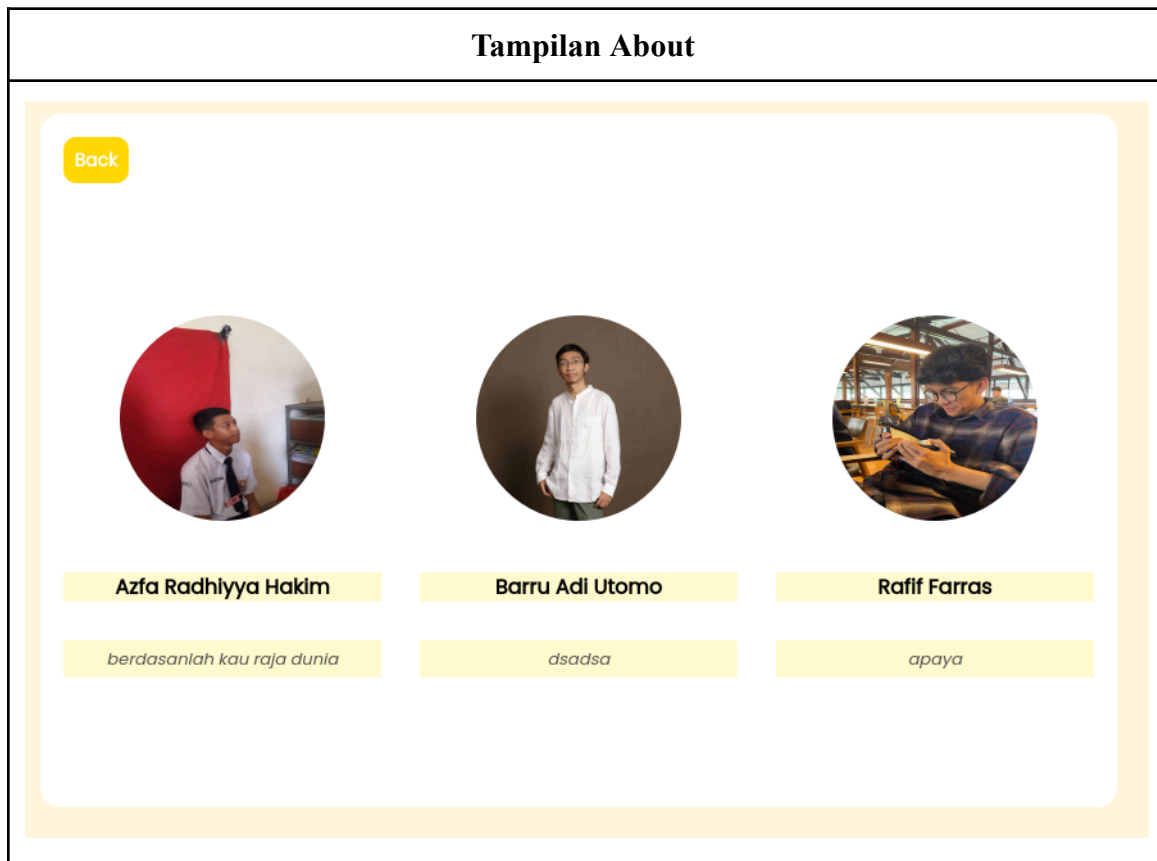
Berikut adalah tampilan awal aplikasi.

Tampilan awal aplikasi



5. Ketika masuk kedalam “CV Analyzer”, di bagian kiri akan ada bar untuk menginput kata kunci yang akan dicari, beserta algoritma yang akan digunakan. Sedangkan “About” berisi informasi terkait pembuat aplikasi ini





6. Pengguna kemudian dapat mengisi input query sesuai dengan kebutuhan, berikut adalah contoh pemakaiannya. Jika dilihat pada bagian result, akan terdapat durasi waktu pencarian.



CV Analyzer App

Keywords:

website

Search time: 23.66 ms

Total: 4 item

Pilih Algoritma:

☒ kmp
☐ booye
☐ aho

Top Matches:

4

Analyze

Result

MOHAMMAD NUGI Total: 5

Matched Keywords:

website: 5 occurrences

Detail View CV

MOHAMMAD NUGI Total: 5

Matched Keywords:

website: 5 occurrences

Detail View CV

MOHAMMAD NUGI Total: 5

Matched Keywords:

website: 5 occurrences

Detail View CV

Mohammad Nuri Total: 2

Matched Keywords:

website: 2 occurrences

Detail View CV

- Pengguna kemudian dapat memilih untuk melihat detail terkait CV yang ditampilkan, yang berisi summary secara keseluruhan. Selain itu pengguna juga dapat melihat file PDF CV secara keseluruhan

Tampilan Detail

Back

View Summary

Biografi Diri

Nama: MOHAMMAD NUGI

Birthdate: 2004-03-22

Address: Jl. Melati No. 45, Bandung

Phone: 082123456781

Skill

- Classes: Social Media Marketing Facebook for Business Project Management Profitability Simulation Nielsen Syndicated Data Analysis
- Excel
- Microsoft Access Microsoft Project Microsoft Dynamics
- Power Presentations Technical Writing
- PowerPoint
- Seminars
- SharePoint BPC and SAP Microsoft Suite: Word

Job History

Position: Product Manager

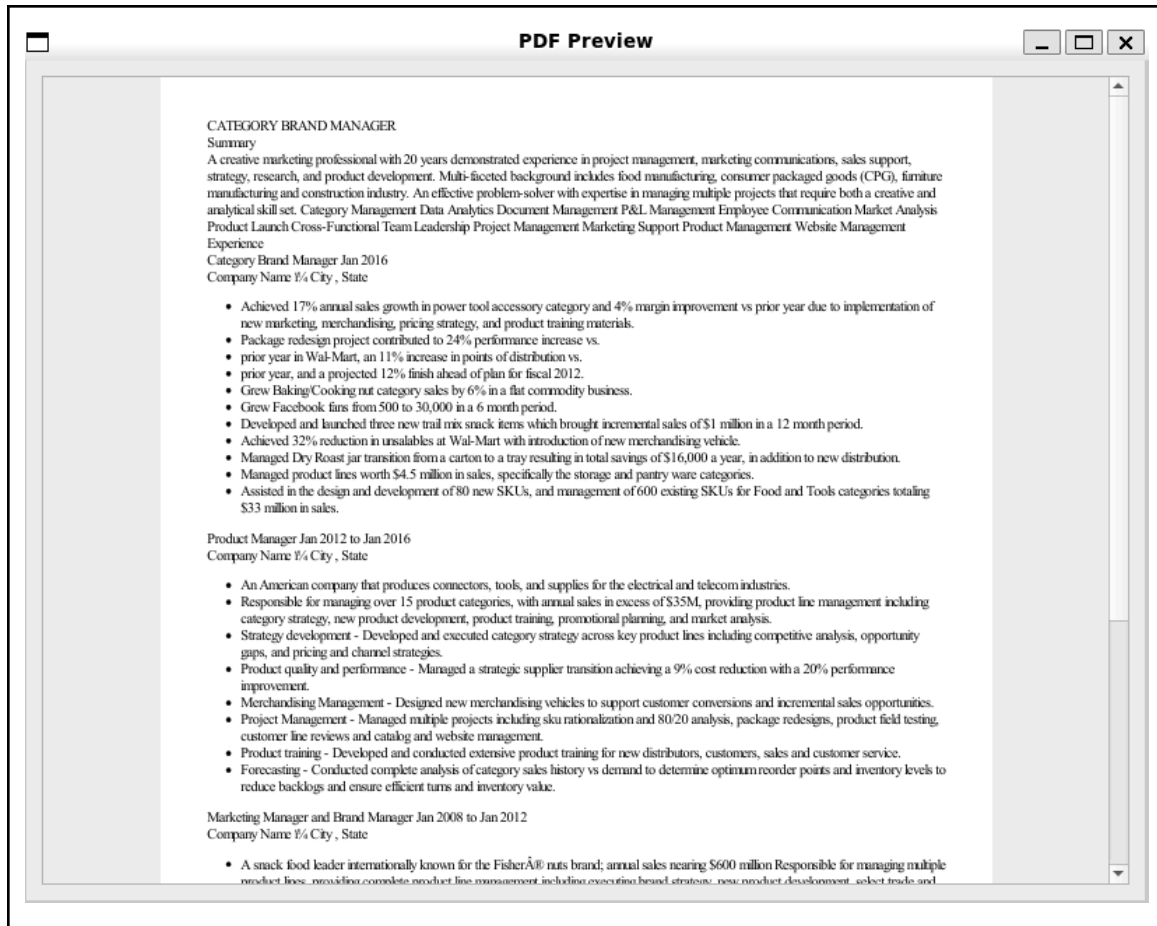
Company: Company Name & City, State

Location:

Description:

- An American company that produces connectors, tools, and supplies for the electrical and telecom industries.
- Responsible for managing over 15 product categories, with annual sales in excess of \$35M, providing product line management including category strategy, new product development, product training, promotional planning, and market analysis.
- Strategy development - Developed and executed category strategy across key product lines including competitive analysis, opportunity gaps, and pricing and channel strategies.
- Product quality and performance - Managed a strategic supplier transition achieving a 9% cost reduction with a 20% performance improvement.
- Merchandising Management - Designed new merchandising vehicles to support customer conversions and incremental sales opportunities.
- Project Management - Managed multiple projects including sku rationalization and 80/20 analysis, package redesigns, product field testing, customer line reviews and catalog and website management.
- Product training - Developed and conducted extensive product training for new distributors, customers, sales and customer service.

Tampilan View CV



4.4. Analisis Hasil Pengujian

Berikut akan kami uji program buatan kami berdasarkan pattern, algoritma, dan banyak CV yang dicari.

Test case 1: “web”, kmp, 4

CV Analyzer App

Keywords:
web

Search time: 538.82 ms

Pilih Algoritma:
☒ kmp
☐ booye
☐ oho

Total: 4 Item

Top Matches:
4

Analyze

Result

R4d3N FRANCISCO Total: 13
Matched Keywords:

web: 13 occurrences

Detail View CV

R4d3N FRANCISCO Total: 13
Matched Keywords:

web: 13 occurrences

Detail View CV

Farh4n N4FIS Total: 12
Matched Keywords:

web: 12 occurrences

Detail View CV

Farh4n N4FIS Total: 12
Matched Keywords:

web: 12 occurrences

Detail View CV

Skill

- Behavioral Management
- Conflict Resolution
- Customer-oriented
- Employee Management
- Employee relations
- Human resources
- Instructional Strategies
- Leadership
- Online Staff Training
- Performance Evaluation
- Project management
- communication skills

Analisis:

Pada test case ini, dilakukan pencarian kata kunci “web” dengan menggunakan algoritma KMP dan batasan 4 hasil teratas. Sesuai dengan gambar, sistem berhasil menemukan 4 item yang cocok dalam waktu pencarian 538.82 ms. Hasil ini menunjukkan bahwa fungsi pencarian berjalan sesuai dengan parameter yang diberikan. Karena keyword “web” ada pada beberapa CV, maka pencarian tidak dilanjutkan dengan fuzzy matching. Jika dilihat pada summary, terlihat bahwa kode berhasil mengekstrak semua skill yang dimiliki applicant.

Test case 2: “Project Management”, boyer moore, 2

CV Analyzer App

Keywords:

Project Management

Pilih Algoritma:

☐ kmp
☒ boaye
☐ aho

Top Matches:

2

Analyze

Search time: 128.08 ms

Total: 2 item

Result

1khw4n 4lH4klm Total: 8

Matched Keywords:

project manic: 8 occurrences

Detail View CV

1khw4n 4lH4klm Total: 8

Matched Keywords:

project manic: 8 occurrences

Detail View CV

Education

Degree: Master

Field: of Arts : Human Resource Development BOWIE STATE UNIVERSITY 1% City , State , US

Institution: N/A

Year: N/A

Location: N/A

Degree: Master

Field: BOWIE STATE UNIVERSITY Bowie, Maryland of Arts in Human Resource Development, May 2012Top [Number]% of

Institution: N/A

Year: N/A

Location: N/A

Analisis:

Pada test case ini, dilakukan pencarian kata kunci “Project Management” dengan menggunakan algoritma Boyer Moore dan batasan 2 hasil teratas. Sesuai dengan gambar, sistem berhasil menemukan 4 item yang cocok dalam waktu pencarian 128ms. Hasil ini menunjukkan bahwa fungsi pencarian berjalan sesuai dengan parameter yang diberikan. Waktu yang diberikan juga relatif lebih singkat dibandingkan algoritma Knuth Morris karena perbandingan dan kompleksitas waktu yang lebih kecil. Jika dilihat pada summary, terlihat bahwa kode berhasil mengekstrak semua education yang dimiliki applicant.

Test case 3: “Linux”, Aho Corasick, 6

CV Analyzer App

Keywords:

Pilih Algoritma:

☐ kmp
☐ boyer
☒ oho

Top Matches:

Analyze

Search time: 255.70 ms

Total: 6 item

Result

4hmad Rafi Total: 4

Matched Keywords:

linux: 4 occurrences

Detail View CV

4hmad Rafi Total: 4

Matched Keywords:

linux: 4 occurrences

Detail View CV

4hmad Rafi Total: 4

Matched Keywords:

linux: 4 occurrences

Detail View CV

4hmad Rafi Total: 4

Matched Keywords:

linux: 4 occurrences

Detail View CV

Moh4mm4d Nu9r Total: 3

Matched Keywords:

linux: 3 occurrences

Detail View CV

Al4hd MuJA Total: 3

Matched Keywords:

linux: 3 occurrences

Detail View CV

PDF Preview

IT CONSULTANT

Professional Summary

Support Engineer with Public Trust Clearance and 10 years of Information Technology experience in installing, maintaining, and repairing hardware, software, and networks. Work well independently, or in a group setting providing all facets of server, computer and network support. Fluent in Spanish.

Core Qualifications

Technical Hardware: Windows, Macintosh, IBM, Dell, Toshiba, Sony, HP, Cisco routers, switches, network printers Operating Systems and Networks: Windows 8.1, Windows 8, Windows 7, Vista, Windows XP, 2000, 98, Windows Server 2008, 2003, Linux, TCP/IP, DNS, DHCP, FTP, VPN; OS X Lion and Mavericks Applications /Software Tools: Microsoft Office 2010, 2007 & 2003, Visio, Adobe Acrobat, Photoshop, Lotus Notes, Symantec Ghost, Symantec PC Anywhere, Dameware, McAfee Antivirus, ImageX, Trend, Putty, WebEX, Westlaw, Parallels Desktop, Microsoft Virtual Server 2005, VM Infrastructure Client, Juniper & Cisco VPN Medical Applications: Casetrakker, eClinicalWorks Databases: Access and MySQL Ticketing Systems: Service Manager, Remedy, Heat, Altiris, Jira

Experience

September 2014 to November 2014

Company Name City , State IT Consultant

- Developed plan for network management platform for release software upgrades for SevOne customers, to include procedures and scripts for backup, high availability continuity of polling during upgrade, and data stitching upon completion.
- Backed up data, restored solutions and data migrations for hardware refresh, cluster migrations and cluster re-alignment projects with customers.
- Actively contributed to the consistent improvement in processes and scripts/automation in delivery of services.
- Troubleshoot issues on Linux servers, Apache Web Server, MySQL, PHP and automation scripts for SevOne clients.

November 2011 to September 2014

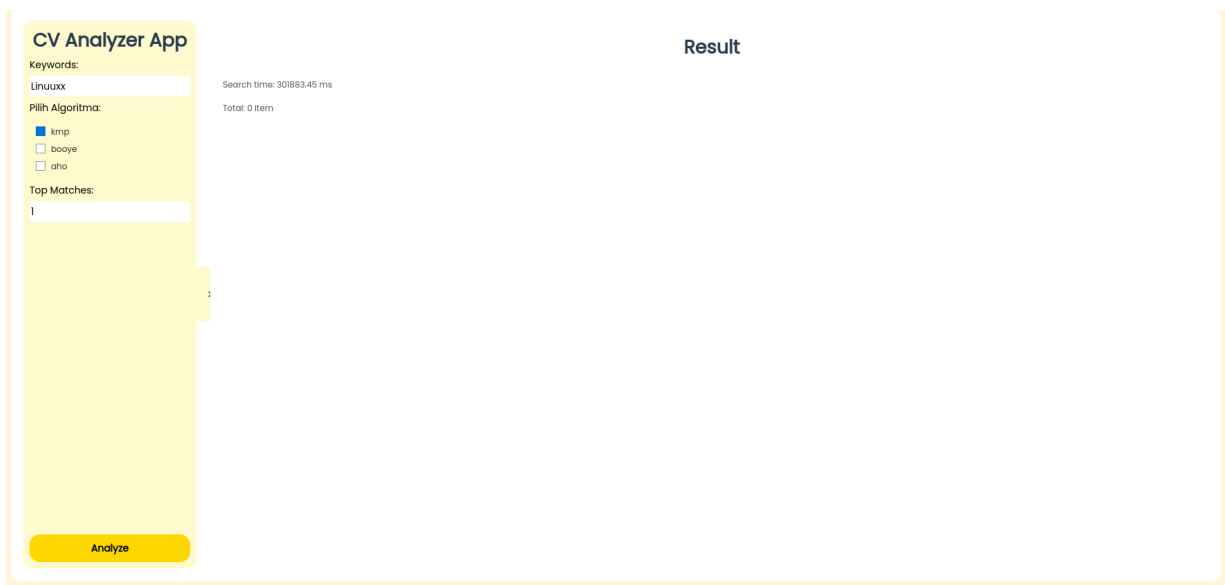
Company Name City , State Systems/Test Engineer

- Contractor for U.S. Immigration and Customs Enforcement at Department of Homeland Security.
- Provided tier 1-3 technical and application support for 60 headquarters staff and 900 remote medical staff users, 2,000 website users with account creation, updating of accounts and issues with website.
- Administered and performed account creation, deletion, permissions and configuration for Medical Applications (Casetrakker and eClinicalWorks) and also related network accounts in Active Directory.
- Tested Casetrakker medical software to identify root causes, verify reported problems or validate and verify resolved issues.
- Setup user accounts, rights, security, systems and network groups with appropriate system and network privileges. Coordinated action with local ITFO's to resolve issues that required escalated issues.
- Performed peripheral and software installations and upgrades on Servers and Desktops including testing of peripheral integration with eClinicalWorks and testing of new eClinicalWorks builds.
- Traveled to field sites to as Lead Engineer for deployment, migration and desktop refreshes to new network.
- Maintained IT hardware and peripheral inventory storage.
- Developed and maintained test cases, create test data and gather results and/or statistics to allow for analysis of issues, leading to satisfactory resolutions.
- Run 3rd party tools such as CDS admin to test workstations and Mid-Tier Diagnostics.
- Tested and provided technical support to local and Field Case Managers users for medical-related applications; eClinicalWorks,

Analysis:

Pada test case ini, dilakukan pencarian kata kunci “Linux” dengan menggunakan algoritma Aho Corasick dan batasan 6 hasil teratas. Sesuai dengan gambar, sistem berhasil menemukan 4 item yang cocok dalam waktu pencarian 215 ms. Hasil ini menunjukkan bahwa fungsi pencarian berjalan sesuai dengan parameter yang diberikan. Waktu yang diberikan juga relatif lebih singkat dibandingkan algoritma Knuth Morris karena perbandingan dan kompleksitas waktu yang lebih kecil. Jika dilihat pada gambar tersebut, aplikasi berhasil menampilkan PDF asli yang dimiliki oleh applicant

Test case 4: “Linuuxxx”, Levenshtein, 1



The screenshot shows the CV Analyzer App interface. On the left, there is a form with the following fields and options:

- Keywords:** A text input field containing "Linuux".
- Pilih Algoritma:** A section with three radio buttons: "kmp" (selected), "booye", and "aho".
- Top Matches:** A text input field containing "1".
- Analyze:** A yellow button at the bottom of the form.

On the right, there is a "Result" section. It displays the search time as "30883.45 ms" and the total number of items as "Total: 0 Item". Below this, there is a large empty space for the results.

Analisis:

Pada test case ini, dilakukan pencarian kata kunci “Linuuxxx” dengan menggunakan algoritma KMP dan batasan 6 hasil teratas. Sesuai dengan gambar, sistem gagal mendapatkan CV yang memiliki keyword tersebut. Dengan demikian, proses dilanjutkan dengan mencari menggunakan algoritma Levenshtein. Dengan waktu yang cukup lama, didapatkan informasi bahwa tidak ada teks yang memiliki persentase kemiripan di atas 75%.

V. PENUTUP

5.1. Kesimpulan

Dalam tugas besar 3 ini, telah berhasil dikembangkan sebuah aplikasi untuk analisis CV dengan mengimplementasikan beberapa algoritma yang relevan, yang dilengkapi dengan GUI untuk visualisasi hasil, dilengkapi dengan enkripsi data untuk keamanan. Pada tugas besar ini telah berhasil diimplementasikan:.

1. Pencocokan String yang menggunakan empat algoritma pencocokan string yang berbeda, yaitu Knuth-Morris-Pratt (KMP), Boyer-Moore, dan Aho-Corasick untuk pencarian kata kunci secara eksak, serta algoritma Levenshtein Distance yang memungkinkan pencocokan fleksibel dengan partial match (*fuzzy matching*) untuk menangani kesalahan pengetikan (*typo*) dengan threshold yang di set sebesar 75%.
2. Sistem mampu mengekstrak bagian-bagian penting dari dokumen CV seperti ringkasan (summary), keahlian (skills), pengalaman (experience), dan pendidikan (education) secara otomatis menggunakan Ekspresi Reguler (Regex) yang dirancang khusus untuk setiap bagian.
3. Keamanan Data: Sebuah mekanisme enkripsi dan dekripsi kustom berhasil dibuat untuk melindungi data teks. Algoritma ini menggunakan pendekatan *hash* berantai yang unik.
4. Antarmuka Pengguna: Seluruh fungsionalitas tersebut diintegrasikan ke dalam sebuah Antarmuka Pengguna Grafis (GUI) yang dibangun menggunakan PyQt5.

5.2. Saran

Untuk pengembangan lebih lanjut dari proyek ini, terdapat beberapa aspek yang dapat di improve, diantaranya pengaplikasian regex yang masih terdapat beberapa kekurangan karena banyaknya kasus/perbedaan format pada PDF. Akan lebih baik jika regex dapat mencakup semua format pdf yang ada agar data tidak null. Akan lebih baik jika memanfaatkan Machine Learning untuk menangani masalah ini. Selanjutnya, bisa ditambahkan berupa benchmarking untuk perbandingan dari setiap algoritma yang ada, agar dapat mengetahui algoritma yang menghasilkan hasil yang paling optimal.

5.3. Refleksi

Pengerjaan tugas besar ini memberikan pemahaman yang mendalam mengenai berbagai algoritma pattern matching yang pastinya akan sangat bermanfaat buat kami di masa depan nanti. Tugas besar ini juga mengajarkan kami untuk berhadapan dengan data CV yang tidak semuanya terstruktur (Berbeda Format) yang mengasah kemampuan kami menghadapi hal tersebut.

Seterusnya, kami juga belajar bagaimana berkomunikasi dan bekerjasama dengan baik melalui tugas besar ini.

LAMPIRAN

Tautan Repository Github

https://github.com/azfaradhi/Tubes3_aaaaa

Tautan Video

<https://youtu.be/CAKHjYiRgAY?feature=shared>

Tabel Kelengkapan Spesifikasi

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

Munir, Rinaldi. 2025. *Pencocokan String (String/Pattern Matching)*.

[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) (Diakses pada 15 Juni 2025).

Munir, Rinaldi. 2025. *String Matching dengan Regular Expression*.

[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) (Diakses pada 15 Mei 2025).