

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

Laporan Tugas Besar 3
IF2211 Strategi Algoritma



Disusun Oleh Kelompok 8:

12821046	Fardhan Indrayesa
13523041	Hanif Kalyana Aditya
13523072	Sabilul Huda

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

2025

DAFTAR ISI

DAFTAR ISI.....	1
Bab 1	
Deskripsi Tugas.....	2
Bab 2	
Landasan Teori.....	7
a. Dasar Teori.....	7
b. Aplikasi Web.....	11
Bab 3	
Analisis Pemecahan Masalah.....	12
a. Langkah-Langkah pemecahan masalah.....	12
b. Proses Pemetaan Masalah (menjadi elemen-elemen KMP dan BM).....	13
c. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun.....	13
d. Contoh Ilustrasi Kasus.....	14
Bab 4	
Implementasi dan Pengujian.....	15
a. Spesifikasi Teknis Program.....	15
b. Tata Cara Penggunaan Program.....	27
c. Hasil Pengujian.....	27
Bab 5	
Kesimpulan dan Saran.....	31
a. Kesimpulan.....	31
b. Saran.....	31
c. Refleksi.....	31
PEMBAGIAN TUGAS.....	32
LAMPIRAN.....	33
DAFTAR PUSTAKA.....	34

Bab 1

Deskripsi Tugas



Gambar 1.1 CV ATS dalam Dunia Kerja
(Sumber: www.cnnindonesia.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.

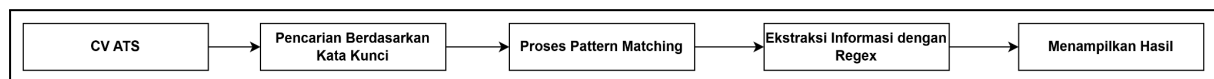
Meskipun demikian, salah satu tantangan besar dalam pengembangan sistem ATS adalah kemampuan untuk memproses dokumen CV dalam format PDF yang tidak selalu terstruktur. Dokumen seperti ini memerlukan metode canggih untuk mengekstrak informasi penting seperti identitas, pengalaman kerja, keahlian, dan 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, penulis diminta untuk mengimplementasikan sistem yang dapat melakukan deteksi informasi pelamar berbasis dokumen CV digital. Metode yang akan digunakan untuk melakukan deteksi pola dalam CV adalah Boyer-Moore dan Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas kandidat melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali profil pelamar secara lengkap hanya dengan menggunakan CV digital.

Penjelasan Implementasi

Dalam tugas ini, penulis akan mengembangkan sebuah sistem ATS (Applicant Tracking System) berbasis CV Digital dengan memanfaatkan teknik Pattern Matching. Implementasi sistem ini akan menggunakan algoritma Boyer-Moore dan Knuth-Morris-Pratt (*Aho-Corasick* apabila mengerjakan bonus) untuk menganalisis dan mencocokkan pola dalam dokumen CV digital, sesuai dengan konsep yang telah dipelajari dalam materi dan slide perkuliahan.




Gambar 1.2 Skema Implementasi *Applicant Tracking System*

Sistem ini bertujuan untuk mencocokkan kata kunci dari user terhadap isi CV pelamar kerja dengan pendekatan pattern matching menggunakan algoritma KMP (Knuth-Morris-Pratt) atau BM (Boyer-Moore). Semua proses dilakukan secara in-memory, tanpa menyimpan hasil pencarian—hanya data mentah (raw) CV yang disimpan. Pengguna (HR atau rekruter) akan memberikan input berupa daftar kata kunci yang ingin dicari (misalnya: "python", "react", dan "sql") serta jumlah CV yang ingin ditampilkan (misalnya Top 10 matches). Setiap file CV dalam format PDF akan dikonversi menjadi satu string panjang yang memuat seluruh teks dari dokumen tersebut. Proses konversi ini bertujuan untuk mempermudah pencocokan pola menggunakan algoritma string matching, sehingga setiap keyword dapat dicari secara efisien dalam satu representasi data linear.

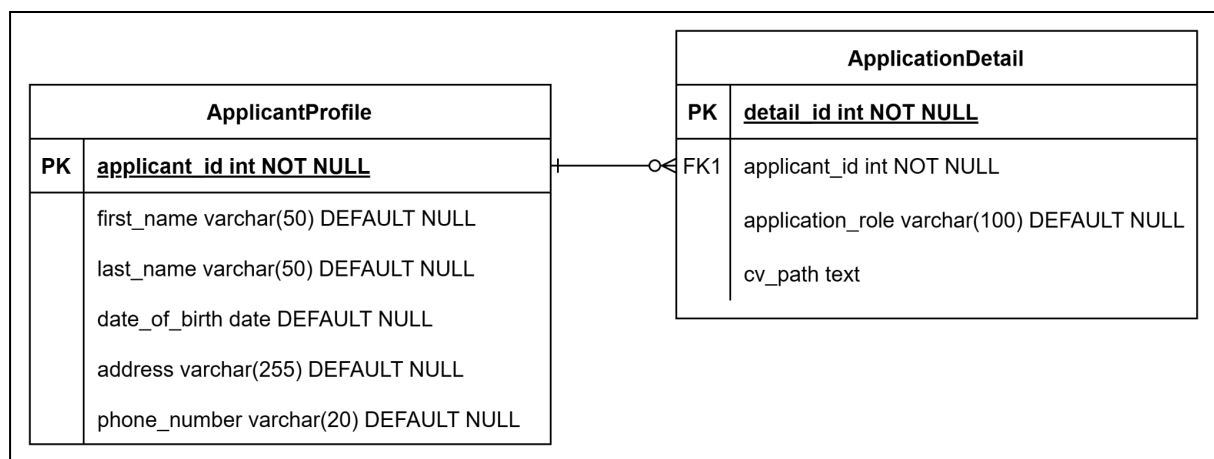
Untuk memberikan pemahaman yang lebih konkret, berikut disajikan contoh kasus penerapan sistem CV ATS beserta prosesnya dan contoh output yang dihasilkan. Dataset yang digunakan dalam contoh ini merupakan dataset CV ATS yang tercantum pada bagian referensi.

Tabel 1.1 Hasil ekstraksi teks dari CV ATS

CV ATS	Ekstraksi Text untuk Regex	Ekstraksi Text untuk <i>Pattern Matching</i> (KMP & BM)
 10276858.pdf	Ekstraksi Text Regex.txt	Ekstraksi Text Pattern Matching.txt

Pada tahap implementasi ini, setiap CV yang telah dikonversi menjadi string panjang untuk mempermudah proses pencocokan. Representasi ini menjadi dasar dalam mencari CV yang

paling relevan dengan kata kunci yang dimasukkan oleh pengguna. Proses pencarian dilakukan dengan menggunakan algoritma pencocokan string Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) untuk menemukan CV yang memiliki kemiripan tertinggi dengan kebutuhan yang ditentukan. Apabila tidak ditemukan satupun CV dalam basis data yang memiliki kecocokan kata kunci secara exact match menggunakan algoritma KMP maupun Boyer-Moore, maka sistem akan mencari CV yang paling mirip berdasarkan tingkat kemiripan di atas ambang batas tertentu (threshold). Hal ini mempertimbangkan kemungkinan adanya kesalahan pengetikan (typo) oleh pengguna atau HR saat memasukkan kata kunci. Penulis diberikan **keleluasaan untuk menentukan nilai ambang batas persentase** kemiripan tersebut, dengan syarat dilakukan pengujian terlebih dahulu untuk menemukan nilai tuning yang optimal dan **dijelaskan secara rinci dalam laporan**. Metode perhitungan tingkat kemiripan harus diterapkan menggunakan algoritma **Levenshtein Distance**.



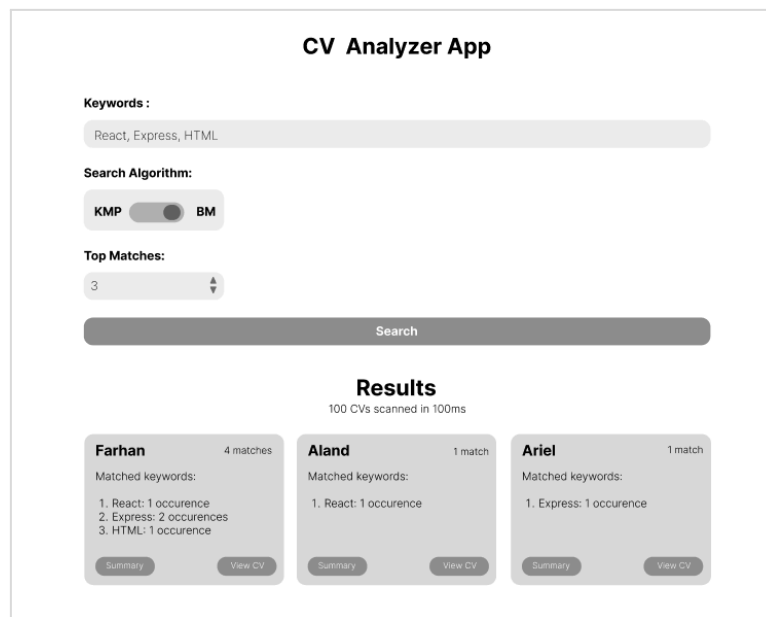
Gambar 1.3 Skema basis data CV ATS

Dalam skema basis data ini, tabel **ApplicantProfile** menyimpan informasi pribadi pelamar, sedangkan tabel **ApplicationDetail** menyimpan detail aplikasi yang diajukan oleh pelamar tersebut. Relasi antara tabel **ApplicantProfile** dan **ApplicationDetail** adalah one-to-many, karena seorang pelamar dapat mengajukan lamaran untuk beberapa posisi dalam perusahaan yang sama, atau bahkan perusahaan yang berbeda. Setiap lamaran mungkin memerlukan dokumen yang berbeda, seperti CV yang telah disesuaikan untuk peran tertentu.

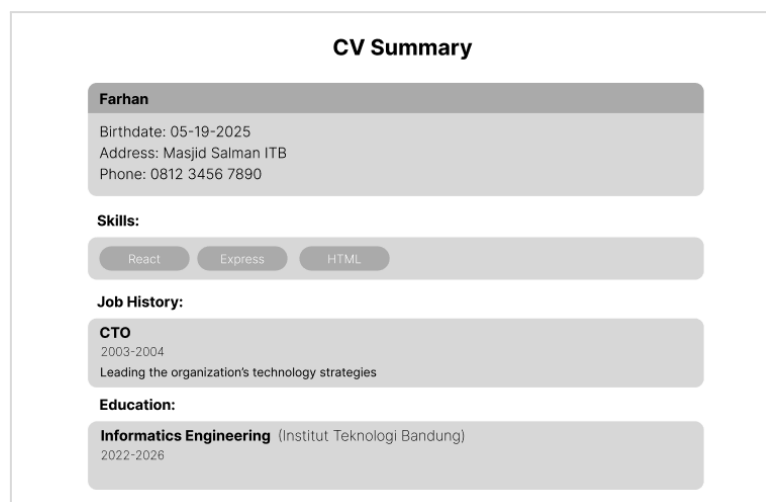
Untuk keperluan pengembangan awal, basis data silahkan **di-seeding secara mandiri** menggunakan data simulasi. Mendekati tenggat waktu pengumpulan tugas, asisten akan menyediakan seeding resmi yang akan digunakan untuk Demo Tugas Besar.

Atribut **cv_path** pada tabel **ApplicationDetail** digunakan untuk menyimpan lokasi berkas CV digital pelamar di dalam repositori sistem. Lokasi penyimpanan mengikuti struktur folder di direktori **data/**, sebagaimana dijelaskan dalam struktur *repository* pada bagian [pengumpulan tugas](#). Berkas CV yang tersimpan akan dianalisis oleh sistem ATS (Applicant Tracking System) yang dikembangkan dalam Tugas Besar ini.

Penggunaan Program



Gambar 1.4 Contoh antarmuka program (Halaman *Home*)



Gambar 1.5 Contoh antarmuka program (Halaman *Summary*)

Penulis diperbolehkan menambahkan elemen tambahan seperti gambar, logo, atau komponen visual lainnya. Desain antarmuka untuk aplikasi desktop **tidak wajib mengikuti tata letak persis** seperti contoh yang diberikan, namun harus dibuat semenarik mungkin, serta tetap mencakup seluruh **komponen wajib yang telah ditentukan**:

- Judul Aplikasi
- Kolom input kata kunci memungkinkan pengguna memasukkan satu atau lebih *keyword*, yang dipisahkan dengan koma, seperti contoh: React, Express, HTML.
- Tombol toggle memungkinkan pengguna memilih salah satu dari dua algoritma pencarian, yaitu KMP atau BM, dengan hanya satu algoritma yang bisa dipilih pada satu waktu.

- *Top Matches Selector* digunakan untuk memilih jumlah CV teratas yang ingin ditampilkan berdasarkan hasil pencocokan.
- *Search Button* digunakan untuk memulai proses pencarian. Diletakkan secara mencolok di bawah *input field*.
- *Summary Result Section* berisi informasi waktu eksekusi pencarian untuk kedua tipe matching yang dilakukan (*exact match* dengan KMP/BM dan *fuzzy match* dengan Levenshtein Distance), misalnya: “Exact Match: 100 CVs scanned in 100ms.\n Fuzzy Match: 100 CVs scanned in 101ms.”
- *Container* hasil pencarian atau kartu CV digunakan untuk menampilkan data hasil pencocokan berdasarkan keyword yang sesuai. Setiap kartu memuat informasi seperti nama kandidat, jumlah kecocokan yang dihitung dari jumlah keyword yang ditemukan, serta daftar kata kunci yang cocok beserta frekuensi kemunculannya. Selain itu, tersedia dua tombol aksi: tombol *Summary* untuk menampilkan ekstraksi informasi dari CV, serta tombol *View CV* yang memungkinkan pengguna melihat langsung file CV asli.

Secara umum, berikut adalah cara umum penggunaan program:

1. Pengguna memasukkan kata kunci pencarian.
2. Memilih algoritma pencocokan: KMP atau BM.
3. Menentukan jumlah hasil yang ingin ditampilkan.
4. Menekan tombol Search.
5. Sistem menampilkan daftar CV yang paling relevan, disertai tombol untuk melihat detail (*Summary*) atau CV asli (*View CV*).

Bab 2

Landasan Teori

a. Dasar Teori

- Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan pola (string) yang memanfaatkan langkah praproses pola dengan membuat array yang dapat menyimpan prefiks terpanjang yang juga merupakan sufiks untuk setiap prefiks pola. Langkah praproses ini memungkinkan algoritma untuk melewati pemeriksaan ulang bagian teks yang telah dicocokkan, sehingga meningkatkan efisiensi. Seperti algoritma brute-force, algoritma KMP ini mengidentifikasi pola pada teks dalam urutan dari kiri ke kanan. Namun, pergeseran pola pada algoritma ini lebih efisien dan terstruktur dari algoritma brute force.

Pada algoritma ini, apabila terjadi ketidakcocokkan setelah membandingkan beberapa karakter, algoritma menggunakan fungsi pinggiran untuk mencegah perbandingan yang tidak perlu dengan menggeser pola pada posisi yang tepat. Berikut merupakan langkah-langkah dalam pencocokan pola menggunakan algoritma KMP.

1. Praproses pola untuk membuat array fungsi pinggiran yang menyimpan prefiks terpanjang yang juga merupakan sufiks pada pola. Array ini digunakan untuk melewati karakter pada teks ketika terjadi ketidakcocokkan.
2. Inisialisasi dua pointer, satu untuk teks dan satu lagi untuk pola, untuk membandingkan karakter.
3. Mulai perbandingan dari karakter pertama dari pola dengan karakter pertama dari teks.
4. Jika karakter cocok, geser kedua pointer ke depan.
5. Jika karakter tidak cocok, gunakan array fungsi pinggiran untuk menentukan seberapa jauh pola dalam pointer harus digeser.
6. Lanjutkan proses ini sampai pola ditemukan atau sampai seluruh teks sudah ditelusuri.

- Algoritma Boyer-Moore

Algoritma Boyer-Moore (BM) adalah algoritma pencocokan pola (*string*) yang sangat efisien karena pendekatannya yang unik, yaitu memulai perbandingan dari karakter paling kanan pada pola, bukan dari kiri. Keunggulan utama algoritma ini terletak pada kemampuannya untuk melakukan pergeseran besar pada pola setiap kali terjadi ketidakcocokkan, sehingga secara signifikan mengurangi jumlah perbandingan yang diperlukan, terutama pada teks yang panjang. Algoritma Boyer-Moore

menggunakan dua aturan atau heuristik utama untuk menentukan seberapa jauh pola harus digeser: *bad-character heuristic* dan *good-suffix heuristic*.

Pada algoritma ini, ketika terjadi ketidakcocokan antara karakter di teks dan pola, algoritma tidak hanya bergeser satu langkah. Sebaliknya, ia menggunakan informasi dari tabel pra-pemrosesan untuk melakukan lompatan cerdas. Berikut merupakan langkah-langkah dalam pencocokan pola menggunakan algoritma Boyer-Moore.

1. Praproses pola untuk membuat tabel *bad-character*. Tabel ini menyimpan posisi kemunculan terakhir dari setiap karakter yang mungkin ada. Informasi ini digunakan untuk menentukan pergeseran maksimum saat karakter yang tidak cocok ditemukan di teks.
2. Sejajarkan pola dengan awal teks. Mulai perbandingan dari karakter paling kanan pada pola dengan karakter yang sesuai di teks.
3. Jika karakter cocok, geser pointer perbandingan ke kiri pada pola dan teks untuk melanjutkan pengecekan (*looking-glass technique*).
4. Jika seluruh karakter pada pola cocok, maka sebuah kecocokan telah ditemukan.
5. Jika karakter tidak cocok, gunakan tabel *bad-character* untuk menentukan seberapa jauh pola harus digeser ke kanan (*character jump technique*). Pergeseran ini didasarkan pada posisi karakter yang tidak cocok di dalam teks.
6. Ulangi proses ini mulai dari langkah 2 sampai seluruh teks sudah ditelusuri.

- Algoritma Aho-Corasick

Algoritma Aho-Corasick merupakan algoritma pencocokan string yang membentuk sebuah mesin status hingga (*finite-state machine*) untuk merepresentasikan semua kata kunci yang ingin dicari. Mesin ini bekerja sebagai algoritma pencocokan berbasis kamus (*dictionary matching*) yang bertujuan mengidentifikasi posisi elemen-elemen dari suatu himpunan string (*dictionary*) di dalam teks masukan. Algoritma ini memiliki tiga komponen utama, yaitu

- goto: Menyimpan setiap node karakter pola dalam bentuk trie.
- failure links: Menyimpan seluruh node yang diikuti oleh karakter lain ketika karakter saat ini tidak memiliki node di dalam trie.
- output: Menyimpan pola pada node yang merupakan akhir karakter dari suatu pola.

Berikut merupakan langkah-langkah dalam pencocokkan pola menggunakan algoritma Aho-Corasick.

1. Bangun trie, diawali dengan membuat node akar.
2. Untuk setiap pola pada list, tambahkan ke trie.
3. Jika sebuah pola berakhir pada suatu node, tambahkan pola tersebut ke node tersebut.

4. Bangun failure links menggunakan BFS untuk menelusuri trie.
5. Untuk setiap node, tetapkan failure link node tersebut ke sufiks terpanjang dari pola saat ini yang juga merupakan prefiks suatu kata kunci dalam trie. Jika tidak ada sufiks yang cocok, tetapkan failure link ke node akar.
6. Dalam melakukan pencarian pola dalam teks, mulai dari node akar trie.
7. Untuk setiap karakter pada teks, ikuti setiap karakter pada trie.
8. Jika pola ditemukan, catat posisinya pada teks.
9. Jika sebuah karakter mengarah ke failure link, ikuti failure link tersebut dan lanjut lakukan pencarian sampai pola ditemukan.

- Algoritma Jarak Levenshtein (Fuzzy Match)

Algoritma Jarak Levenshtein adalah sebuah metrik yang digunakan untuk mengukur "jarak" atau perbedaan antara dua urutan string. Dalam konteks pencarian, algoritma ini menjadi dasar dari fuzzy matching, yaitu sebuah teknik untuk menemukan teks yang "mirip" meskipun tidak sama persis. Hal ini sangat berguna untuk mengatasi kesalahan pengetikan (typo) atau variasi kata. Jarak ini dihitung berdasarkan jumlah minimum operasi penyuntingan satu karakter (insersi, delesi, atau substitusi) yang diperlukan untuk mengubah satu string menjadi string yang lain.

Dalam aplikasi ini, implementasi fuzzy matching diperluas untuk menangani dua skenario berbeda. Jika kata kunci yang dicari adalah sebuah kata tunggal, maka Jarak Levenshtein dihitung antara kata kunci tersebut dengan setiap kata unik di dalam teks CV. Jika hasilnya di bawah ambang batas (threshold), maka dianggap sebagai kecocokan. Namun, jika kata kunci adalah sebuah frasa (mengandung spasi), pendekatannya berbeda. Frasa tersebut dipecah menjadi kata-kata individual, dan sistem akan memeriksa apakah setiap kata individual tersebut dapat ditemukan secara "mirip" di dalam kumpulan kata-kata unik dari teks CV. Sebuah frasa hanya akan dianggap cocok jika semua komponen katanya berhasil ditemukan. Berikut merupakan langkah-langkah konseptual dalam perhitungan Jarak Levenshtein.

1. Buat sebuah matriks (tabel 2D) dengan dimensi (panjang *string* pertama + 1) x (panjang *string* kedua + 1).
2. Inisialisasi baris pertama dengan nilai 0, 1, 2, ... dan kolom pertama dengan nilai yang sama. Nilai ini merepresentasikan biaya untuk mengubah *string* kosong menjadi prefiks dari *string* lain.
3. Iterasi melalui setiap sel di dalam matriks, mulai dari (1,1). Untuk setiap sel, bandingkan karakter yang sesuai dari kedua *string*.
4. Tentukan biaya substitusi: 0 jika karakter sama, atau 1 jika berbeda.
5. Isi nilai sel saat ini dengan nilai minimum dari: nilai sel di atas + 1 (delesi), nilai sel di kiri + 1 (insersi), atau nilai sel di kiri-atas + biaya substitusi.
6. Lanjutkan proses ini hingga seluruh matriks terisi. Nilai pada sel paling kanan bawah matriks adalah Jarak Levenshtein antara kedua *string* tersebut.

- Enkripsi Data

Enkripsi secara fundamental adalah proses mengubah data dari format yang dapat dibaca manusia (disebut plaintext) menjadi format yang tidak dapat dipahami dan acak (disebut ciphertext). Tujuan utama dari enkripsi adalah untuk menjaga kerahasiaan (confidentiality) informasi, memastikan bahwa hanya pihak yang memiliki wewenang yang dapat mengakses dan memahami data asli. Proses ini dicapai dengan menggunakan sebuah algoritma kriptografi dan sebuah kunci (key). Algoritma adalah serangkaian aturan atau prosedur matematis untuk melakukan transformasi data, sedangkan kunci adalah sepotong informasi rahasia yang mengontrol cara kerja algoritma. Tanpa kunci yang benar, ciphertext akan sangat sulit atau bahkan mustahil untuk dikembalikan ke bentuk plaintext-nya.

Dalam aplikasi ini, enkripsi diterapkan untuk melindungi data pribadi pelamar yang disimpan di dalam database MySQL. Informasi sensitif seperti nama, tanggal lahir, alamat, dan nomor telepon diubah menjadi format terenkripsi sebelum disimpan. Proses kebalikannya, yaitu dekripsi, dilakukan saat aplikasi perlu menampilkan data tersebut kepada pengguna. Metode ini memastikan bahwa bahkan jika seseorang berhasil mendapatkan akses langsung ke isi database, mereka tidak akan dapat membaca informasi pribadi pelamar tanpa memiliki kunci dekripsi yang benar, sehingga menambahkan lapisan keamanan yang signifikan pada sistem.

- Metode Stream Cipher dengan XOR dan LCG

Metode enkripsi yang diimplementasikan dalam proyek ini adalah stream cipher sederhana. Berbeda dengan block cipher yang mengenkripsi data dalam blok-blok berukuran tetap, stream cipher bekerja dengan mengenkripsi data secara kontinu, satu per satu, baik itu per bit maupun per byte. Pendekatan ini sangat cocok untuk data yang ukurannya bervariasi, seperti nama atau alamat. Inti dari implementasi ini adalah operasi logika XOR (exclusive OR). XOR memiliki sifat simetris yang elegan: jika $A \text{ XOR } B = C$, maka $C \text{ XOR } B = A$. Artinya, operasi dan kunci yang sama dapat digunakan untuk proses enkripsi dan dekripsi.

Kekuatan enkripsi berbasis XOR sepenuhnya bergantung pada kualitas aliran data acak yang digunakan sebagai kunci, yang disebut key stream. Untuk menghasilkan key stream ini, kita menggunakan Linear Congruential Generator (LCG), sebuah algoritma klasik untuk menghasilkan serangkaian angka pseudo-acak. Sebuah kunci rahasia (secret key) yang telah ditentukan di dalam kode pertama-tama diubah menjadi nilai awal atau seed. Seed ini kemudian dimasukkan ke dalam LCG untuk menghasilkan deretan angka yang unik namun deterministik. Setiap byte dari plaintext kemudian di-XOR dengan setiap byte dari key stream yang dihasilkan oleh LCG. Karena key stream yang sama dapat dibangkitkan kembali menggunakan secret key yang sama, proses dekripsi dapat dilakukan dengan mudah dengan melakukan operasi XOR sekali lagi pada ciphertext.

b. Aplikasi Web

Arsitektur sistem yang dibangun untuk aplikasi CV Analyzer ini mengadopsi pendekatan modular dengan menerapkan pola tiga lapis (three-tier architecture) yang memisahkan antara lapisan presentasi (UI), lapisan logika bisnis (Core), dan lapisan data. Untuk meningkatkan portabilitas dan keamanan, sistem ini juga dilengkapi dengan mekanisme konfigurasi eksternal. Pemisahan ini bertujuan untuk meningkatkan keterbacaan kode, mempermudah pemeliharaan, dan memungkinkan pengembangan setiap komponen secara independen, memastikan setiap bagian memiliki tanggung jawab yang jelas dan terdefinisi.

Lapisan pertama adalah Presentation Layer atau Antarmuka Pengguna (UI), yang sepenuhnya bertanggung jawab untuk menampilkan informasi kepada pengguna dan menangkap input dari mereka. Dibangun menggunakan framework CustomTkinter, lapisan ini terdiri dari jendela utama (`main_window.py`) yang menampilkan kontrol pencarian dan hasil, serta jendela sekunder (`summary_window.py`) untuk menampilkan detail ringkasan CV. Komponen-komponen UI yang dapat digunakan kembali, seperti kartu CV dan dialog peringatan, diisolasi dalam modulnya sendiri (`widgets.py`). Lapisan ini tidak mengandung logika pemrosesan data, tugasnya murni sebagai jembatan visual antara pengguna dan fungsionalitas inti aplikasi.

Lapisan kedua, yang merupakan inti dari sistem, adalah Logic Layer atau Core. Lapisan ini berfungsi sebagai otak dari aplikasi. Modul `search_handler.py` bertindak sebagai orkestrator utama yang menerima permintaan dari UI. Ia kemudian memanggil fungsi spesifik dari modul `algorithms.py`, yang berisi implementasi murni dari algoritma pencocokan pola (KMP, BM, Aho-Corasick), algoritma fuzzy matching Jarak Levenshtein, dan ekstraksi Regex. Sebagai bagian dari fitur bonus, lapisan ini juga mencakup `encryption_handler.py` yang mengelola semua operasi kriptografi menggunakan metode stream cipher sederhana. Arsitektur ini juga diperkuat dengan `config_manager.py`, sebuah utilitas yang membaca kredensial dari file `config.ini` eksternal, sehingga tidak ada informasi sensitif yang disimpan secara hardcoded di dalam kode.

Lapisan terakhir adalah Data Layer, yang mengelola semua aspek persistensi dan pengambilan data. Interaksi dengan database MySQL sepenuhnya dikelola oleh modul `db_connector.py`, yang mengabstraksikan semua perintah SQL. Berkat `config_manager`, modul ini dapat terhubung ke database tanpa memerlukan kredensial internal. Data profil pelamar di dalam database kini disimpan dalam format BLOB yang terenkripsi, memastikan kerahasiaan informasi. Selain database, lapisan ini juga mencakup sistem berkas tempat semua dokumen CV dalam format PDF disimpan secara fisik, menyediakan sumber data mentah yang akan diproses oleh lapisan logika.

Bab 3

Analisis Pemecahan Masalah

a. Langkah-Langkah pemecahan masalah

Secara umum, alur pemecahan masalah yang diterapkan oleh aplikasi untuk menemukan kandidat yang relevan berdasarkan kata kunci dapat diuraikan ke dalam beberapa langkah utama. Langkah-langkah ini dirancang untuk memastikan proses pencarian berjalan secara efisien dan mampu menangani berbagai skenario, termasuk kesalahan pengetikan.

1. Inisialisasi dan Pengambilan Input: Aplikasi menerima input dari pengguna berupa daftar kata kunci (*keywords*), algoritma pencocokan eksak yang akan digunakan (KMP, BM, atau AC), dan jumlah hasil teratas (*top matches*) yang ingin ditampilkan.
2. Pengambilan dan Persiapan Data: Sistem mengambil seluruh data profil pelamar beserta path absolut ke file CV mereka dari *database* MySQL. Untuk setiap pelamar, konten dari file CV dibaca dan dinormalisasi menjadi satu *string* panjang dalam format huruf kecil.
3. Pencocokan Eksak (*Exact Matching*): Untuk setiap CV, sistem melakukan pencarian untuk setiap kata kunci yang diberikan menggunakan algoritma pencocokan pola yang telah dipilih oleh pengguna. Proses ini bertujuan untuk menemukan kecocokan yang sempurna dan menghitung frekuensi kemunculannya.
4. Pencocokan Kabur (*Fuzzy Matching*) Bersyarat: Setelah pencocokan eksak selesai, sistem akan memeriksa apakah ada kata kunci yang belum ditemukan dan apakah jumlah kandidat yang cocok masih kurang dari *top matches* yang diminta. Jika kondisi terpenuhi, sistem akan menjalankan pencocokan kabur untuk setiap kata kunci yang belum ditemukan. Logika ini secara cerdas membedakan antara kata kunci tunggal dan frasa. Untuk kata kunci tunggal, pencarian kemiripan dilakukan menggunakan Jarak Levenshtein. Untuk frasa, sistem menggunakan metode *sliding window of words* untuk memeriksa apakah urutan kata yang mirip dengan frasa tersebut ada di dalam CV.
5. Perhitungan Skor dan Pemeringkatan: Setiap kandidat diberikan skor berdasarkan hasil dari kedua tahap pencocokan. Skor ini dihitung berdasarkan jumlah kata kunci unik yang cocok dan total frekuensi kemunculan. Selanjutnya, semua kandidat diurutkan dari skor tertinggi ke terendah.
6. Penyajian Hasil: Aplikasi mengambil kandidat teratas sesuai dengan jumlah *top matches* yang diminta dan menampilkannya kepada pengguna dalam bentuk kartu ringkasan.
7. Ekstraksi Detail Sesuai Permintaan (*On-Demand*): Ketika pengguna memilih untuk melihat ringkasan detail dari seorang kandidat, sistem akan membaca kembali file CV kandidat tersebut dan menjalankan serangkaian *Regular*

Expression (Regex) untuk mengekstrak informasi spesifik seperti riwayat pekerjaan, pendidikan, dan keahlian untuk ditampilkan di jendela *summary*.

b. Proses Pemetaan Masalah (menjadi elemen-elemen KMP dan BM)

Pemetaan masalah dalam konteks ini adalah proses menerjemahkan kebutuhan fungsional sistem—yaitu menemukan kandidat berdasarkan CV—ke dalam terminologi dan komponen yang dapat diproses oleh algoritma. Setiap algoritma yang digunakan memiliki cara pemetaan yang spesifik. Untuk algoritma pencocokan pola seperti Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick (AC), masalah utama adalah menemukan kemunculan sebuah "pola" di dalam sebuah "teks". Dalam aplikasi ini, pola dipetakan sebagai setiap kata kunci (*keyword*) yang dimasukkan oleh pengguna (misalnya, "python"), sedangkan teks adalah keseluruhan konten dari satu file CV yang telah dinormalisasi. Komponen penting lainnya adalah tabel pra-pemrosesan. Untuk KMP, ini adalah *array Longest Proper Prefix* (LPS) yang menyimpan informasi tentang sufiks yang juga merupakan prefiks. Untuk BM, komponen ini adalah tabel *bad-character* yang menyimpan posisi terakhir setiap karakter untuk memungkinkan lompatan cerdas. Sementara untuk Aho-Corasick, komponennya adalah struktur data *Trie* yang dikombinasikan dengan *failure links* untuk mencari semua kata kunci secara simultan.

Untuk *fuzzy matching*, masalahnya kini dipecah menjadi dua kasus yang ditangani secara berbeda. Jika pola adalah sebuah kata tunggal dengan potensi kesalahan ketik, masalah ini dipetakan ke dalam konsep jarak sunting (*edit distance*) antara kata kunci dan setiap kata unik di dalam teks. Namun, jika pola adalah sebuah frasa, masalahnya dipetakan ke dalam pencarian urutan kata yang mirip. Sistem akan memecah frasa menjadi beberapa kata, lalu mencari kemunculan urutan kata yang sama di dalam teks CV, di mana setiap kata dalam urutan tersebut diperbolehkan memiliki sedikit perbedaan (kesalahan ketik) yang diukur dengan Jarak Levenshtein. Ini memastikan bahwa baik kata kunci "java developer" maupun "java deeloper" dapat ditemukan, tetapi CV yang hanya mengandung kata "java" saja tidak akan dianggap sebagai kecocokan untuk frasa tersebut.

c. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

Arsitektur aplikasi secara fungsional dirancang untuk mendukung alur pemrosesan algoritma secara efisien. Ketika pengguna berinteraksi dengan antarmuka di *Presentation Layer*, misalnya dengan memilih "BM" dari *combo box* dan menekan tombol "Search", permintaan tersebut tidak langsung dieksekusi oleh UI. Sebaliknya, informasi input (kata kunci, algoritma "BM", jumlah hasil) diteruskan ke *search_handler.py* di dalam *Logic Layer*. Modul ini bertindak sebagai pusat kendali yang pertama-tama memanggil *db_connector.py* untuk mengambil data,

kemudian berdasarkan pilihan "BM", ia secara spesifik memanggil fungsi `boyer_moore_search()` dari dalam `algorithms.py` untuk setiap CV. Jika diperlukan, ia juga akan memanggil `find_fuzzy_matches()` dari modul yang sama. Setelah semua hasil dikumpulkan dan diurutkan, `search_handler` mengemasnya menjadi format yang standar dan mengembalikannya ke UI, yang kemudian bertugas untuk menampilkannya secara visual.

d. Contoh Ilustrasi Kasus

Sebagai ilustrasi, bayangkan seorang perekrut bernama Budi ingin mencari kandidat pengembang perangkat lunak. Ia membuka aplikasi CV Analyzer dan memasukkan tiga kata kunci ke dalam kolom input: "java, spring, postgresl" (terdapat kesalahan ketik pada kata terakhir). Ia memilih algoritma "BM" dan mengatur "Top Matches" menjadi 10, lalu menekan tombol "Search". Aplikasi pertama-tama akan mengambil data dari 480 CV yang ada di *database*. Selanjutnya, `search_handler` akan memulai siklus pencocokan eksak. Menggunakan algoritma Boyer-Moore, ia menemukan banyak CV yang mengandung kata "java" dan "spring", namun tidak ada satupun yang mengandung kata "postgresl" secara persis.

Setelah siklus pertama selesai, program mendeteksi bahwa jumlah kandidat yang cocok (misalnya, baru 5 kandidat) masih di bawah permintaan Budi (10), dan masih ada satu kata kunci ("postgresl") yang belum ditemukan. Oleh karena itu, sistem secara otomatis memulai siklus *fuzzy matching*. Ia mengambil kata "postgresl" dan membandingkannya dengan kata-kata di dalam CV menggunakan Jarak Levenshtein. Sistem menemukan bahwa kata "postgresql" memiliki jarak Levenshtein 1 dari "postgresl", yang berada di bawah ambang batas toleransi (misalnya 2). Akibatnya, beberapa kandidat yang memiliki "postgresql" kini juga mendapatkan skor tambahan. Setelah semua proses selesai, aplikasi menggabungkan semua skor, mengurutkan kandidat, dan menampilkan 10 kandidat teratas kepada Budi. Ketika Budi mengklik tombol "Summary" pada salah satu kandidat, aplikasi akan membaca ulang file PDF kandidat tersebut, menjalankan fungsi ekstraksi Regex untuk mendapatkan detail riwayat pekerjaan dan pendidikan, lalu menampilkannya di jendela ringkasan.

Bab 4

Implementasi dan Pengujian

a. Spesifikasi Teknis Program

- Knuth-Morris-Pratt (KMP)

Berikut merupakan pseudocode dari algoritma Knuth-Morris-Pratt

```
FUNCTION kmp_search(text, pattern):
    N = length of text
    M = length of pattern
    lps_array = create array of size M

    compute_lps_array(pattern, M, lps_array)

    i = 0 // pointer untuk text
    j = 0 // pointer untuk pattern
    count = 0

    WHILE i < N:
        IF pattern[j] == text[i]:
            i = i + 1
            j = j + 1

        IF j == M:
            count = count + 1
            j = lps_array[j - 1]
        ELSE IF i < N AND pattern[j] != text[i]:
            IF j != 0:
                j = lps_array[j - 1]
            ELSE:
                i = i + 1

    RETURN count
END FUNCTION

PROCEDURE compute_lps_array(pattern, M, lps_array):
    length = 0
    lps_array[0] = 0
    i = 1

    WHILE i < M:
        IF pattern[i] == pattern[length]:
            length = length + 1
            lps_array[i] = length
            i = i + 1
        ELSE:
            IF length != 0:
                length = lps_array[length - 1]
            ELSE:
                lps_array[i] = 0
                i = i + 1
    END PROCEDURE
```

Berdasarkan pseudocode di atas, berikut merupakan penjelasan dari masing-masing fungsi dan prosedur yang digunakan

Class	
Implementasi bersifat prosedural dan terdiri dari fungsi-fungsi mandiri yang berada di dalam modul <code>algorithms.py</code> .	
Attribute	
-	
Fungsi dan Prosedur	
<code>kmp_search(text, pattern)</code>	Fungsi utama yang bertugas mengorkestrasi proses pencarian pola. Fungsi ini menerima teks dan pola sebagai input, memanggil prosedur <code>compute_lps_array</code> untuk pra-pemrosesan, kemudian melakukan pencocokan, dan mengembalikan jumlah total kemunculan pola di dalam teks.
<code>compute_lps_array(pat, M, lps)</code>	Sebuah prosedur pembantu (<i>helper procedure</i>) yang melakukan tahap pra-pemrosesan pada pola. Tugasnya adalah membangun <i>array Longest Proper Prefix</i> (LPS) yang juga merupakan sufiks. <i>Array</i> ini krusial untuk menentukan seberapa jauh pergeseran pola harus dilakukan ketika terjadi ketidakcocokan, sehingga meningkatkan efisiensi algoritma secara keseluruhan.

- Boyer-Moore (BM)

Berikut merupakan pseudocode dari algoritma Boyer-Moore

```

FUNCTION boyer_moore_search(text, pattern):
    m = length of pattern
    n = length of text
    bad_char_table = build_bad_char_table(pattern, m)

    count = 0
    current_pos = 0

    WHILE current_pos <= n - m:
        match_pos = find_first_match(text from current_pos, pattern,
        bad_char_table)

        IF match_pos != -1:
            count = count + 1
            current_pos = current_pos + match_pos + 1

```

```

ELSE:
    BREAK // Tidak ada lagi kecocokan

RETURN count
END FUNCTION

FUNCTION find_first_match(text_segment, pattern, bad_char_table):
    m = length of pattern
    n = length of text_segment
    i = m - 1
    j = m - 1

    WHILE i < n:
        IF pattern[j] == text_segment[i]:
            IF j == 0:
                RETURN i // Kecocokan ditemukan di indeks i
            ELSE:
                i = i - 1
                j = j - 1
        ELSE:
            last_occurrence = get from bad_char_table for character
            text_segment[i]
            i = i + m - min(j, 1 + last_occurrence)
            j = m - 1

    RETURN -1 // Tidak ada kecocokan
END FUNCTION

```

Berdasarkan pseudocode di atas, berikut merupakan penjelasan dari masing-masing fungsi dan prosedur yang digunakan

Class	
Implementasi bersifat prosedural dan terdiri dari fungsi-fungsi mandiri yang berada di dalam modul <code>algorithms.py</code> .	
Attribute	
-	
Fungsi dan Prosedur	
<code>boyer_moore_search(text, pattern)</code>	Fungsi utama yang mengoordinasikan pencarian. Fungsi ini membangun tabel <i>bad-character</i> , lalu secara berulang memanggil <code>find_first_match</code> pada sisa teks hingga seluruh teks selesai ditelusuri, dan mengembalikan jumlah total kecocokan.
<code>bad_char_heuristic(string, size)</code>	Sebuah prosedur pembantu yang melakukan pra-pemrosesan pada pola untuk membangun

	tabel <i>bad-character</i> . Tabel ini, yang diimplementasikan sebagai <i>dictionary</i> , menyimpan indeks kemunculan terakhir dari setiap karakter di dalam pola.
<code>find_first_match_in_segment(.. ..)</code>	Fungsi inti yang mengimplementasikan logika pencarian Boyer-Moore untuk menemukan kecocokan pertama dalam sebuah segmen teks. Fungsi ini menerapkan teknik <i>looking-glass</i> (mundur saat cocok) dan <i>character jump</i> (melompat saat tidak cocok).

- Aho-Corasick (AC)

Berikut merupakan pseudocode dari algoritma Aho-Corasick

```

CLASS TrieNode:
  CONSTRUCTOR:
    children = empty map
    output = empty list
    failure = null
END CLASS

CLASS AhoCorasick:
  CONSTRUCTOR(text, keywords):
    this.text = text
    this.keywords = to_lowercase(keywords)
    build_trie()
    build_failure_links()

  PROCEDURE build_trie():
    root = new TrieNode()

    FOR each keyword IN keywords:
      current_node = root
      FOR each character ch IN keyword:
        IF ch NOT IN current_node.children:
          current_node.children[ch] = new TrieNode()
          current_node = current_node.children[ch]
        APPEND keyword TO current_node.output
    END PROCEDURE

  PROCEDURE build_failure_links():
    queue = empty queue
    root.failure = root

    FOR each child_node IN root.children:
      child_node.failure = root
      ENQUEUE child_node INTO queue

    WHILE queue NOT EMPTY:
      current_node = DEQUEUE from queue

```

```

    FOR each character ch, child_node IN current_node.children:
        fail_node = current_node.failure

        WHILE fail_node ≠ root AND ch NOT IN fail_node.children:
            fail_node = fail_node.failure

        IF ch IN fail_node.children:
            child_node.failure = fail_node.children[ch]
        ELSE:
            child_node.failure = root

        child_node.output = child_node.output +
child_node.failure.output
        ENQUEUE child_node INTO queue
    END PROCEDURE

FUNCTION search():
    current_node = root
    found = empty list

    FOR i FROM 0 TO length of text - 1:
        ch = text[i]

        WHILE current_node ≠ root AND ch NOT IN current_node.children:
            current_node = current_node.failure

        IF ch IN current_node.children:
            current_node = current_node.children[ch]

        FOR each matched_keyword IN current_node.output:
            start_index = i - length(matched_keyword) + 1
            APPEND (start_index, matched_keyword) TO found

    found_count = empty map

    IF found NOT EMPTY:
        FOR each keyword IN keywords:
            count = 0
            FOR each (index, kw) IN found:
                IF kw == keyword:
                    count = count + 1
            found_count[keyword] = count
        ELSE:
            FOR each keyword IN keywords:
                found_count[keyword] = 0

    RETURN found_count
END FUNCTION
END CLASS

```

Berdasarkan pseudocode di atas, berikut merupakan penjelasan dari masing-masing fungsi dan prosedur yang digunakan

trie_node

Class

<code>trie_node</code>	Kelas node untuk membentuk struktur trie. Setiap node menyimpan child node, daftar output (pola yang cocok), dan failure link.
Attribute	
<code>children</code>	Dictionary yang menyimpan karakter sebagai key dan <code>trie_node</code> sebagai value, mewakili child dari node tersebut.
<code>output</code>	List yang menyimpan pola yang cocok di node tersebut.
<code>failure</code>	Penunjuk ke node failure jika tidak ada karakter yang cocok. Digunakan saat pencarian gagal di suatu jalur.

`aho_corasick`

Class	
<code>aho_corasick</code>	Kelas utama yang mengimplementasikan algoritma Aho-Corasick untuk pencocokan banyak pola sekaligus dalam teks.
Attribute	
<code>kws</code>	List yang berisi pola yang akan dicocokkan di dalam teks, diubah menjadi huruf kecil,
<code>text</code>	Teks utama (string) tempat pencarian dilakukan.
<code>root</code>	Objek root dari trie, sebagai titik awal automaton.
Fungsi dan Prosedur	
<code>build_trie</code>	Fungsi untuk membangun struktur trie dari daftar pola. Setiap karakter dari pola ditambahkan ke node trie.
<code>build_failure</code>	Fungsi untuk membangun failure link untuk setiap node menggunakan algoritma BFS. Failure link ini digunakan saat pencarian

	ketika terdapat karakter yang tidak cocok.
search	Fungsi untuk melakukan pencarian semua pola dalam teks berdasarkan trie dan failure link yang sudah dibuat. Mengembalikan jumlah kemunculan setiap pola.

- Jarak Levenshtein (Fuzzy Match)

Berikut merupakan pseudocode dari algoritma Fuzzy Match

```

FUNCTION fuzzy_search(text, pattern, threshold):
  IF pattern contains space:
    RETURN _find_fuzzy_phrase(text, pattern, threshold)
  ELSE:
    RETURN _find_fuzzy_single_word(text, pattern, threshold)
END FUNCTION

FUNCTION _find_fuzzy_single_word(text, pattern, threshold):
  count = 0
  unique_words_in_text = get all unique words from text
  FOR each word in unique_words_in_text:
    IF absolute(length of word - length of pattern) <= threshold:
      distance = levenshtein_distance(word, pattern)
      IF distance <= threshold:
        count = count + 1
  RETURN count
END FUNCTION

FUNCTION _find_fuzzy_phrase(text, phrase, threshold):
  pattern_words = split phrase into words
  text_words = split text into words
  count = 0
  m = length of pattern_words
  n = length of text_words

  FOR i from 0 to n - m:
    window = get a slice of text_words from i to i + m
    total_distance = 0
    is_match = TRUE

    FOR j from 0 to m - 1:
      dist = levenshtein_distance(window[j], pattern_words[j])
      IF dist > threshold:
        is_match = FALSE
        BREAK
      total_distance = total_distance + dist

    IF is_match AND (total_distance / m) <= threshold:
      count = count + 1

  RETURN count
END FUNCTION

```

Berdasarkan pseudocode di atas, berikut merupakan penjelasan dari masing-masing fungsi dan prosedur yang digunakan

Class	
Implementasi bersifat prosedural dan terdiri dari fungsi-fungsi mandiri yang berada di dalam modul <code>algorithms.py</code> .	
Attribute	
-	
Fungsi dan Prosedur	
<code>fuzzy_search(text, pattern, threshold)</code>	Fungsi penyalur (dispatcher) utama untuk pencocokan kabur. Fungsi ini menganalisis pattern untuk mendeteksi adanya spasi. Berdasarkan hasil analisis, ia akan secara otomatis memanggil fungsi internal yang tepat, yaitu <code>_find_fuzzy_single_word</code> atau <code>_find_fuzzy_phrase</code> .
<code>_find_fuzzy_single_word(...)</code>	Fungsi internal yang khusus menangani pencocokan untuk kata kunci tunggal. Fungsi ini membandingkan pattern dengan setiap kata unik di dalam text menggunakan Jarak Levenshtein.
<code>_find_fuzzy_phrase(...)</code>	Fungsi internal yang khusus menangani pencocokan untuk frasa . Fungsi ini menerapkan metode "jendela geser kata-kata" (<i>sliding window of words</i>) untuk mencari urutan kata yang mirip di dalam text , dengan memberikan toleransi kesalahan ketik pada setiap kata dalam urutan tersebut.
<code>levenshtein_distance(s1, s2)</code>	Fungsi inti yang mengimplementasikan algoritma Jarak Levenshtein menggunakan pendekatan pemrograman dinamis. Fungsi ini mengembalikan sebuah nilai integer yang merepresentasikan jarak sunting antara s1 dan s2.

- *Regular Expression* (Regex)

Berikut merupakan pseudocode untuk mendapatkan informasi CV berdasarkan pola Regex yang sudah ditetapkan

```
def extract_details_with_regex(full_cv_text):
    text = re.sub(r' +', ' ', full_cv_text).strip()
    text = re.sub(r'([a-z])([A-Z])', r'\1 \2', text) # Tambahkan spasi
    antara camelCase jika ada
    details = {"summary": "", "skills": [], "job_history": [],
    "education": []}

    # --- Ekstrak Summary ---
    summary_match =
re.search(r'summary\s*(.*?) (?=highlights|skills|experience|education|wor
k history)', text, re.IGNORECASE)
    if summary_match:
        details["summary"] = summary_match.group(1).strip()

    # --- Ekstrak Skills (ambil blok saja tanpa dipecah) ---
    # skills_match =
re.search(r'(skills|highlights)\s*(.*?) (?=experience|education|summary|w
ork history)', text, re.IGNORECASE)
    if not details["skills"]:
        fallback_skills_match =
re.search(r'(?::accomplishments|strengths|capabilities)\s*(.*?) (?=experie
nce|education|summary|work history)', text, re.IGNORECASE)
        if fallback_skills_match:
            block = fallback_skills_match.group(1).strip()
            details["skills"] = [block] if block else []

    # --- Ekstrak Job History ---
    job_matches = re.findall(
r'(\d{2}/\d{4})\s*(?:to|-)?\s*\d{2}/\d{4}|\d{6}|\w+\s+\d{4}\s*(?:to|-)?\s
*\w+\s+\d{4})\s*company
name\s*city\s*,?\s*state\s*(.*?)\s*(?=\d{2}/\d{4}|company
name|education|$)',
        text,
        re.IGNORECASE
    )
    for date_str, after_text in job_matches:
        if re.match(r'\d{6}', date_str):
            month = date_str[:2]
            year = date_str[2:]
            date_str = f"{month}/{year}"
            title_match = re.search(r'([A-Za-z\s/&-]{3,})',
after_text.strip())
            # title_match = re.search(r'(?::Company
Name\s*(?)[A-Z][a-zA-Z\s/&-]+?) (?=\s*\d{2}/\d{4}|\s*\b\w+\s+\d{4})',
entry, re.IGNORECASE)
            # title_match = re.search(r'(chef|cook|line cook|food service
cook|supervisor|prep chef|server)', after_text, re.IGNORECASE)
            title = title_match.group(1).title() if title_match else
"Unknown"
            details["job_history"].append({
                "title": title,
                "company": "Company Name",
                "dates": date_str
            })
```



```

# --- Ekstrak Education ---
edu_match = re.search(r'education\s*(.*)', text, re.IGNORECASE)
if edu_match:
    block = edu_match.group(1).strip()
    date_match = re.search(r'(\d{4})', block)
    # Ekstrak major setelah kata "diploma:" atau "degree in"
    # major_match =
re.search(r'(:diploma\s*|degree\s*(?:in)?|courses\s*in|major\s*in)?\s*
([a-zA-Z ,&-\s])+', block, re.IGNORECASE)
    major_match = re.search(r'(:diploma\s*:\s*|degree
in\s*) ([a-zA-Z ,&-\s])+', block, re.IGNORECASE)
    # Ambil nama institusi sebelum diploma
    institution_match =
re.search(r'([A-Z][\w\s,&-\s]+(?:university|institute|college|academy|sch
ool|polytechnic|center|centre|faculty|campus))', block, re.IGNORECASE)
    # institution_match =
re.search(r'(:(:at|from|in)?\s*) ([A-Z][\w\s,&-\s]+)?,\s*(?:\b(?:univer
sity|institute|college|academy|school|polytechnic|center|centre|faculty|
campus)\b)?', block, re.IGNORECASE)
    # institution_match = re.search(r'(.*)\s+(?:high
school|diploma|degree)', block, re.IGNORECASE)

    if date_match:
        details["education"].append({
            "major": major_match.group(1).strip() if major_match
else "",
            "institution": institution_match.group(1).strip() if
institution_match else "",
            "dates": date_match.group(1)
        })

    return details

```

Berdasarkan pseudocode di atas, berikut merupakan penjelasan dari masing-masing fungsi dan prosedur yang digunakan

Class	
Implementasi bersifat prosedural dan terdiri dari fungsi-fungsi mandiri yang berada di dalam modul algorithms.py.	
Attribute	
-	
Fungsi dan Prosedur	
extract_details_with_regex(full_cv_text)	Fungsi untuk melakukan ekstraksi dari string yang telah dilakukan pattern matching. Ekstraksi yang dimaksud adalah melakukan pengambilan informasi tertentu dari cv antara lain <i>summary</i> , <i>skills</i> , <i>job history</i> , dan

	<i>education</i> . Nantinya informasi ini akan ditampilkan di <code>summary_window</code> .
--	---

- Enkripsi (Stream Cipher dengan LCG)

Berikut merupakan pseudocode dari algoritma

```

CLASS LCG:
    state
    FUNCTION __init__(seed):
        self.state = seed
    FUNCTION next_byte():
        self.state = (A * self.state + C) mod M
        RETURN 8 bit teratas dari self.state

FUNCTION encrypt(plain_text):
    seed = generate_seed_from_key(SECRET_KEY)
    lcg = new LCG(seed)
    plain_bytes = convert plain_text to bytes
    encrypted_bytes = empty byte array

    FOR each byte in plain_bytes:
        keystream_byte = lcg.next_byte()
        encrypted_byte = byte XOR keystream_byte
        append encrypted_byte to encrypted_bytes

    RETURN encrypted_bytes
END FUNCTION

// Fungsi dekripsi mengikuti logika yang sama persis dengan fungsi
enkripsi

```

Berdasarkan pseudocode di atas, berikut merupakan penjelasan dari masing-masing fungsi dan prosedur yang digunakan

Class	
LCG (Linear Congruential Generator)	Kelas ini bertanggung jawab untuk membangkitkan serangkaian angka pseudo-acak (<i>key stream</i>) yang digunakan untuk proses enkripsi dan dekripsi.
Attribute	
state	Atribut internal dari kelas LCG yang

	menyimpan nilai saat ini dalam deret angka pseudo-acak. Nilai ini terus diperbarui setiap kali <code>next_byte()</code> dipanggil.
<code>SECRET_KEY</code>	Sebuah konstanta <i>string</i> yang berfungsi sebagai kunci rahasia utama. Kunci ini diubah menjadi <i>seed</i> numerik untuk memastikan bahwa <i>key stream</i> yang dihasilkan bersifat deterministik dan dapat direplikasi untuk proses dekripsi.
Fungsi dan Prosedur	
<code>LCG.__init__(seed)</code>	Prosedur konstruktor untuk kelas LCG. Ia menginisialisasi generator dengan sebuah nilai awal (<i>seed</i>) yang didapat dari <code>SECRET_KEY</code> .
<code>LCG.next_byte()</code>	Fungsi yang mengimplementasikan rumus LCG untuk menghasilkan byte pseudo-acak berikutnya dalam key stream.
<code>_process_data(input_bytes, key_str)</code>	Fungsi internal inti yang melakukan operasi XOR antara setiap byte dari data masukan dengan setiap byte dari key stream yang dihasilkan oleh LCG.
<code>encrypt(plain_text)</code>	Fungsi publik yang menerima teks biasa, mengubahnya menjadi byte array, dan memanggil <code>_process_data</code> untuk menghasilkan data biner terenkripsi.
<code>decrypt(encrypted_bytes)</code>	Fungsi publik yang menerima data biner terenkripsi, memanggil <code>_process_data</code> (yang logikanya sama persis karena sifat simetris XOR) untuk mengembalikannya ke bentuk byte array asli, lalu mengubahnya kembali menjadi string teks biasa.

- Proses Enkripsi pada Saat Pemuatan Data

Proses enkripsi data dirancang untuk mengamankan informasi pribadi pelamar pada saat data tersebut disimpan (at rest) di dalam database MySQL. Proses ini dieksekusi secara eksklusif oleh skrip `load_data.py` saat pertama kali mengisi database. Untuk setiap data profil pelamar yang dihasilkan, sebelum dikirim ke database, setiap kolom data yang

bersifat sensitif seperti nama, tanggal lahir, alamat, dan nomor telepon, akan dilewatkan ke fungsi encrypt yang berada dalam modul encryption_handler.py. Di dalam fungsi ini, data teks biasa (plaintext) pertama-tama diubah menjadi format biner (bytes), kemudian setiap byte dari data tersebut dikenai operasi logika XOR dengan sebuah byte dari aliran kunci (key stream). Aliran kunci ini dibangkitkan secara pseudo-acak oleh sebuah Linear Congruential Generator (LCG) yang diinisialisasi menggunakan sebuah SECRET_KEY internal. Hasil akhir dari proses ini adalah data biner terenkripsi (ciphertext) yang kemudian disimpan ke dalam kolom bertipe BLOB di database, memastikan bahwa data yang tersimpan tidak dapat dibaca secara langsung.

- Proses Dekripsi pada Saat Program Utama Berjalan

Proses dekripsi terjadi secara dinamis setiap kali pengguna melakukan pencarian dan aplikasi utama perlu menampilkan data pelamar. Proses ini dimulai ketika modul search_handler.py memanggil fungsi get_all_applicants_with_cv() dari db_connector.py. Setelah db_connector berhasil mengambil baris-baris data dari database—yang mana masih dalam format BLOB terenkripsi—ia tidak langsung mengembalikannya. Sebaliknya, untuk setiap baris, setiap kolom data terenkripsi dilewatkan ke fungsi decrypt() dari modul encryption_handler.py. Kunci keberhasilan proses ini terletak pada sifat simetris dari operasi XOR. Fungsi decrypt() menggunakan SECRET_KEY yang sama persis dengan yang digunakan saat enkripsi untuk membangkitkan aliran kunci (key stream) yang identik. Ketika ciphertext dikenai operasi XOR dengan aliran kunci yang sama, hasilnya adalah data biner asli (plaintext), yang kemudian diubah kembali menjadi format string yang dapat dibaca. Dengan demikian, program utama "mengetahui" kuncinya karena baik proses enkripsi maupun dekripsi sama-sama mengimpor dan menggunakan konstanta SECRET_KEY dari satu modul terpusat yang sama, yaitu encryption_handler.py, sehingga menjamin konsistensi dan integritas operasi kriptografi.

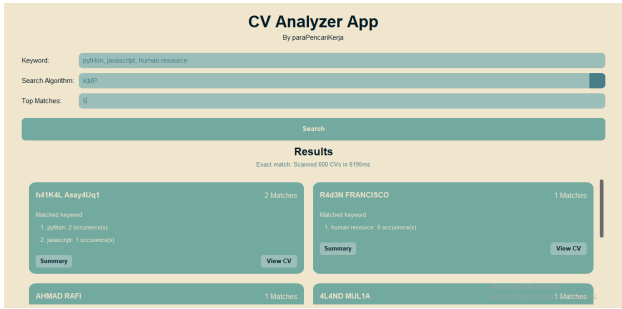

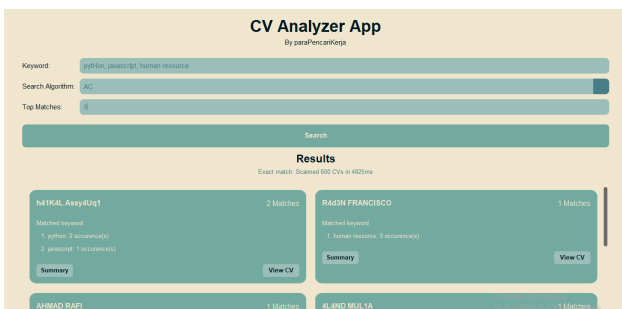
b. Tata Cara Penggunaan Program

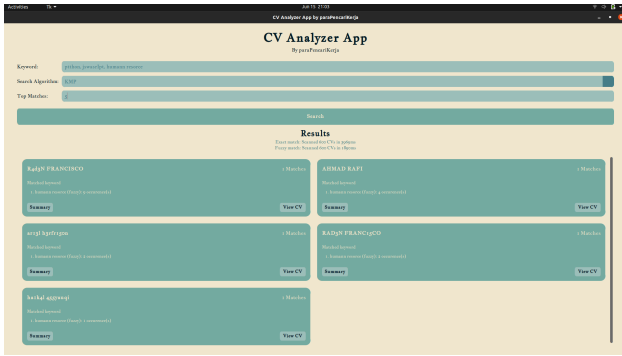
Berikut merupakan tata cara penggunaan program.

1. Masuk ke direktori project /Tubes3_ParaPencariKerja
2. Pada direktori yang sama, edit file config.ini, lalu sesuaikan host, user, dan password untuk server database MySQL.
3. Pada direktori /Tubes3_ParaPencariKerja/src, jalankan main.py.
4. Tulis pola (keyword) pada kolom Keyword, contoh "React, python, market analysis" tanpa tanda kutip.
5. Pilih algoritma pencarian pola, terdapat algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), atau Aho-Corasick (AC).
6. Tetapkan banyaknya maksimal CV yang ingin ditampilkan pada kolom Top Matches.
7. Klik tombol Search untuk mulai lakukan pencarian.

8. Setelah pencarian selesai, apabila terdapat CV yang cocok, akan ditampilkan nama aplikasi, banyaknya kata kunci (pola) yang muncul dalam CV tersebut, banyaknya kata kunci yang ditemukan, tombol Summary, dan tombol View CV.
9. Tombol View CV akan menampilkan CV aplikasi yang dibuka di browser.
10. Tombol Summary akan menampilkan rangkuman isi CV dari aplikasi tersebut, yang menunjukkan informasi pribadi, skill, pengalaman, dan riwayat pendidikan.

c. Hasil Pengujian

Algoritma	Konfigurasi	Hasil
Knuth-Morris-Pratt (KMP)	Keywords: pytHon, javascRipt, human resource Top matches: 5	 <p>Banyak CV ditemukan: 5 Waktu: 6195 ms</p>
Boyer-Moore (BM)	Keywords: pytHon, javascRipt, human resource Top matches: 5	 <p>Banyak CV ditemukan: 5 Waktu: 4882 ms</p>
Aho-Corasick (AC)	Keywords: pytHon, javascRipt, human resource Top matches: 5	 <p>Banyak CV ditemukan: 5 Waktu: 4825 ms</p>

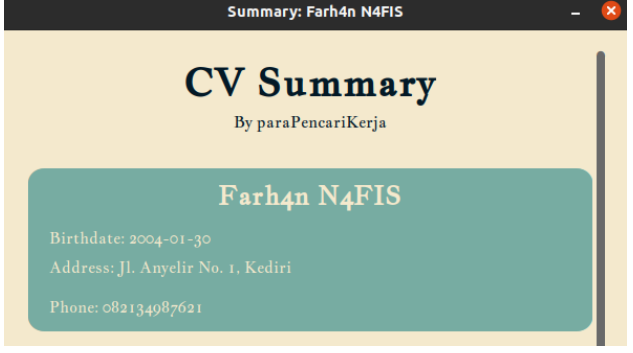
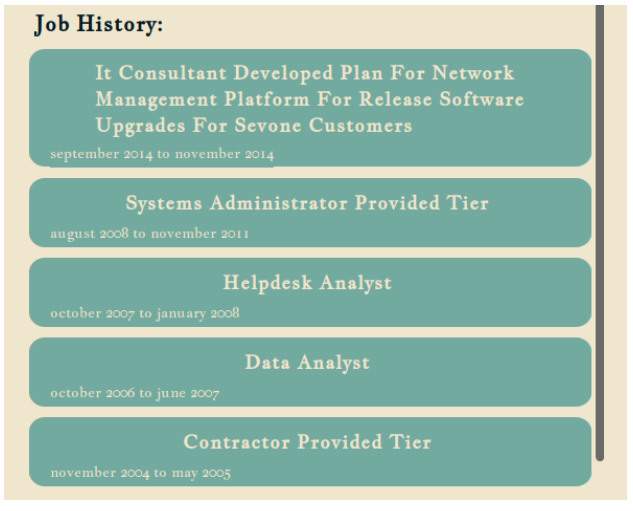
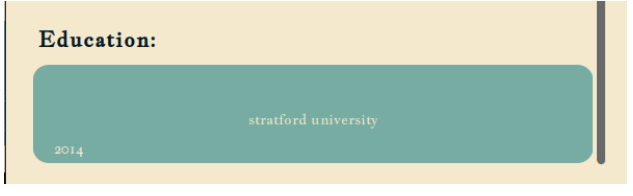
<i>Fuzzy match</i>	Keywords: ptthon, jswasrIpt, humann resorce Top matches: 5	 <p> Banyak CV ditemukan: 5 Waktu (fuzzy): 1890 ms Waktu (exact AC): 3969 ms </p>
--------------------	---	---

Berdasarkan hasil pada ketiga algoritma pencarian pola dalam teks pada tabel di atas, diperoleh bahwa waktu yang dibutuhkan untuk masing-masing algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick (AC) dalam mencari tiga pola berbeda dalam teks adalah 6195 ms, 4882 ms, dan 4825 ms. Berdasarkan hasil tersebut, didapatkan bahwa algoritma AC merupakan algoritma yang paling unggul (memiliki waktu pencarian tercepat). Pada algoritma KMP, terdapat pre-process pola terlebih dahulu, yaitu membuat array/tabel fungsi pinggiran yang mempunyai kompleksitas waktu $O(m)$ dan kompleksitas waktu pencarian string $O(n)$, dengan m adalah panjang pola dan n adalah panjang teks, sehingga kompleksitas waktu algoritma KMP adalah $O(m+n)$ untuk setiap pola dan $O(kn)$ untuk banyak pola, dengan k adalah banyaknya pola. Algoritma ini membandingkan seluruh teks terhadap pola, sehingga waktu yang dibutuhkan lebih lama.

Pada algoritma BM, digunakan pre-process pembuatan array/tabel last occurrence function pada heuristik bad character, yaitu tabel yang menyimpan indeks terakhir kemunculan setiap karakter yang unik pada pola. Untuk kasus terburuk pada satu pola, kompleksitas waktu algoritma BM adalah $O(nm)$ dengan n adalah panjang teks dan m adalah panjang pola. Sedangkan pada banyak pola, kompleksitas algoritmanya sama seperti KMP, yaitu $O(kn)$. Namun, karena algoritma ini memanfaatkan beberapa heuristik seperti bad character, tidak semua karakter pada teks dibandingkan dengan pola. Hal ini menyebabkan algoritma ini memiliki waktu yang lebih singkat daripada algoritma KMP.

Algoritma Aho-Corasick (AC) dirancang khusus untuk pencarian banyak pola secara simultan dalam sebuah teks. Algoritma ini menggunakan struktur data *finite state automaton* berupa trie (pohon prefix) yang dilengkapi dengan *failure links*, memungkinkan pencarian efisien seperti halnya dalam KMP namun untuk banyak pola sekaligus. Setiap pola akan dibangun dalam bentuk path dari akar ke daun dalam trie. Jika pencarian gagal pada suatu cabang, *failure link* akan memindahkan pencarian ke simpul *fallback* yang paling cocok. Kompleksitas waktu algoritma AC untuk banyak pola adalah $O(n + z)$, dengan n merupakan panjang teks dan z merupakan total kemunculan semua pola dalam teks. Algoritma ini memproses teks hanya sekali untuk seluruh pola, sehingga lebih unggul daripada algoritma KMP dan BM dalam pencarian banyak pola.

Algoritma pencocokan kabur (*fuzzy matching*) yang diimplementasikan menggunakan Jarak Levenshtein berjalan dalam skenario yang berbeda dan secara fundamental lebih kompleks daripada algoritma pencocokan eksak. Ia hanya dieksekusi secara kondisional ketika pencarian eksak tidak memberikan hasil yang cukup, dengan tujuan mengukur "kemiripan" antarkata untuk mengatasi kesalahan pengetikan. Secara teknis, algoritma ini menghitung jumlah minimum operasi penyuntingan (insersi, delesi, atau substitusi) untuk mengubah satu *string* menjadi *string* lain dengan kompleksitas waktu $O(p \cdot q)$, menjadikannya operasi yang intensif. Untuk menangani berbagai jenis input, implementasinya dirancang secara cerdas menggunakan sebuah fungsi penyalur (*dispatcher*) yang dapat membedakan antara kata kunci tunggal dan frasa. Jika input adalah frasa, maka digunakan metode "jendela geser kata-kata" (*sliding window of words*) untuk mencari urutan kata yang mirip, memastikan bahwa hasil pencarian tetap relevan dan akurat meskipun beban komputasinya lebih berat.

Algoritma	Konfigurasi	Hasil
Regular Expression	Overview: (ekspresi)	
	Job History: (ekspresi)	
	Education: (ekspresi)	

	Skills: (ekspresi)	<div>Skills:</div> <div>served on a tiger team which identified and resolved general ledger postings</div>
--	-----------------------	--

Pada hasil summary menggunakan Regex

Bab 5

Kesimpulan dan Saran

a. Kesimpulan

Pada tugas ini, penulis telah berhasil membuat aplikasi pencarian CV berdasarkan pola yang diberikan. Algoritma yang digunakan untuk proses pencarian tersebut adalah algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), Aho-Corasick (AC), dan fuzzy match. Semua algoritma tersebut dapat menghasilkan jumlah kemunculan yang sama, dengan pendekatan dan waktu yang berbeda. Algoritma KMP memanfaatkan fungsi pinggiran yang menunjukkan prefiks pola terpanjang yang juga merupakan sufiks dari pola tersebut. Algoritma BM menggunakan heuristik seperti bad character, yang dapat melewati beberapa karakter dalam teks, sehingga pencariannya menjadi lebih cepat. Algoritma AC memanfaatkan finite automaton dalam melakukan pencarian banyak pola pada teks, sehingga lebih cepat daripada algoritma KMP dan BM. Fuzzy match digunakan apabila exact match tidak berhasil menemukan pola yang cocok dalam teks. Selain itu, digunakan juga regular expression dalam mencari beberapa informasi dalam CV, seperti overview, skil, education, dan pengalaman. Tugas ini diselesaikan dengan menggunakan bahasa pemrograman Python dengan penyimpanan data berbasis MySQL.

b. Saran

Saran untuk eksperimen ini adalah sebagai berikut.

- (1) Menambahkan analisis performa memori pada setiap algoritma
- (2) Dapat menambahkan analisis untuk data CV yang berbeda bahasa.
- (3) Menggunakan pendekatan modern berbasis vektor (seperti word embedding dan cosine similarity)

c. Refleksi

Refleksi yang kami peroleh dari tugas ini adalah pentingnya manajemen waktu dan komunikasi tim dalam menyelesaikan proyek berskala besar. Kami menyadari bahwa memulai pengerjaan lebih awal akan memberikan waktu yang cukup untuk mengevaluasi dan mengoptimasi hasil kerja, baik dari sisi frontend maupun backend. Melalui tugas ini, kami mendapatkan pengalaman dalam mengembangkan aplikasi pencarian CV berdasarkan keyword yang diberikan. Kami juga belajar mengenai implementasi algoritma pencocokan string seperti KMP, BM, AC, dan fuzzy match, serta regular expression.

PEMBAGIAN TUGAS

NIM	Nama	Pembagian Tugas
12821046	Fardhan Indrayesa	<p>Implementasi</p> <ul style="list-style-type: none"> - Load data - Aho-Corasick <p>Laporan</p> <ul style="list-style-type: none"> - Deskripsi tugas - Dasar teori: Algoritma Aho-Corasick - Spesifikasi: Aho-Corasick - Analisis KMP, BM, dan AC - Tata cara penggunaan program - Kesimpulan
13523041	Hanif Kalyana Aditya	<p>Implementasi</p> <ul style="list-style-type: none"> - Regular Expression <p>Laporan</p> <ul style="list-style-type: none"> - Spesifikasi: Regular Expression - Analisis: Regular Expression
13523072	Sabilul Huda	<p>Implementasi</p> <ul style="list-style-type: none"> - Algoritma KMP, BM - Algoritma Fuzzy Matching - Enkripsi dan dekripsi data dengan metode Stream Cipher dengan XOR dan LCG - Design dan implementasi UI menggunakan tkinter - Konfigurasi username dan password untuk mysql - Perancangan arsitektur utama program <p>Laporan</p> <ul style="list-style-type: none"> - Dasar teori: algoritma KMP, algoritma BM, algoritma jarak levenshtein (fuzzy match) - Penjelasan aplikasi - Analisis Fuzzy Match - Spesifikasi KMP, BM, fuzzy match, enkripsi

LAMPIRAN

Tautan *repository* GitHub : [Tubes3_ParaPencariKerja](#)

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

DAFTAR PUSTAKA

- Aho-Corasick algorithm*. Aho-Corasick algorithm - Algorithms for Competitive Programming. (2025, April 18). https://cp-algorithms.com/string/aho_corasick.html
- GeeksforGeeks. (n.d.). *KMP (Knuth-Morris-Pratt) Algorithm for Pattern Searching in C*. <https://www.geeksforgeeks.org/c/kmp-algorithm-for-pattern-searching-in-c/>
- GeeksforGeeks. (n.d.). *Aho-Corasick Algorithm for Pattern Searching*. [Aho-Corasick Algorithm for Pattern Searching - GeeksforGeeks](https://www.geeksforgeeks.org/aho-corasick-algorithm-for-pattern-searching/)
- Munir, R. (2025). *Pencocokan string (string matching) dengan algoritma brute force, KMP, Boyer-Moore* [Lecture slides]. Program Studi Teknik Informatika, STEI ITB. [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)
- Munir, R. (2025). *Pencocokan string dengan regular expression (regex)* [Lecture slides]. Program Studi Teknik Informatika, STEI ITB. [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)
- Boyer-Moore Algorithm. (n.d.). *GeeksforGeeks*. <https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/>
- Cloudflare. (n.d.). *What is encryption?*. <https://www.cloudflare.com/learning/ssl/what-is-encryption/>
- Microsoft. (2023, Juni 21). *N-tier architecture style*. Microsoft Learn. <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier>
- Oracle. (2025). *MySQL 8.0 Reference Manual*. <https://dev.mysql.com/doc/refman/8.0/en/>
- Python Software Foundation. (2025). *Python 3.12.4 documentation*. <https://docs.python.org/3/>
- Python Software Foundation. (2025). *re — Regular expression operations*. <https://docs.python.org/3/library/re.html>
- Python Software Foundation. (2025). *tkinter — Python interface to Tcl/Tk*. <https://docs.python.org/3/library/tkinter.html>
- TutorialsPoint. (n.d.). *Stream Cipher - XOR, LCG*. https://www.tutorialspoint.com/cryptography/stream_cipher.htm
- Wikipedia. (2025, Mei 29). *Linear congruential generator*. https://en.wikipedia.org/wiki/Linear_congruential_generator