

**Laporan Tugas Besar 3**  
**IF2211 Strategi Algoritma**  
**Pemanfaatan *Pattern Matching* untuk Membangun**  
**Sistem ATS (Applicant Tracking System) Berbasis CV Digital**  
**Semester II Tahun 2024/2025**



Disusun oleh :  
Clarissa Nethania Tambunan (13523016)  
Shannon Aurellius Anastasya Lie (13523019)  
Andrew Isra Saputra DB (13523110)

Dosen Pengampu:  
Dr. Nur Ulfa Maulidevi, S.T, M.Sc.  
Dr. Ir. Rinaldi Munir, M.T.  
Menterico Adrian, S.T, M.T.

**Program Studi Teknik Informatika**  
**Sekolah Teknik Elektro Dan Informatika**  
**Institut Teknologi Bandung**  
**2025**

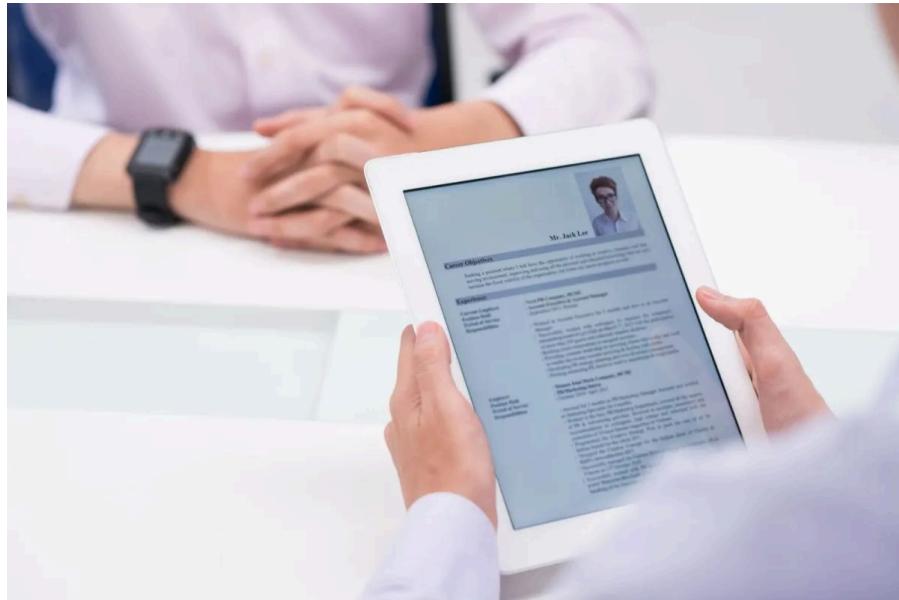
## DAFTAR ISI

<b>BAB I DESKRIPSI TUGAS.....</b>	<b>4</b>
<b>BAB II LANDASAN TEORI.....</b>	<b>6</b>
2.1. Algoritma Knuth-Morris-Pratt (KMP).....	6
2.2. Algoritma Boyer Moore (BM).....	6
2.3. Algoritma Aho-Corasick.....	7
2.4. Levenshtein Distance.....	7
2.5. Sistem ATS (Applicant Tracking System).....	8
2.6. Curriculum Vitae (CV).....	8
2.7. Penjelasan Mengenai Aplikasi Web yang Dibangun.....	8
<b>BAB III ANALISIS PEMECAHAN MASALAH.....</b>	<b>9</b>
3.1. Langkah Pemecahan Masalah.....	9
3.2. Proses Pemetaan Masalah.....	9
3.4. Ilustrasi Kasus.....	14
<b>BAB IV IMPLEMENTASI DAN PENGUJIAN.....</b>	<b>16</b>
4.1. Spesifikasi Teknis Program.....	16
4.1.1. Struktur Program ATS.....	16
4.1.2. File aho_corasick.py di folder src/core/.....	17
4.1.3. File boyer_moore.py di folder src/core/.....	18
4.1.4. File encryption.py di folder src/core/.....	18
4.1.5. File kmp.py di folder src/core/.....	19
4.1.6. File levenshtein.py di folder src/core/.....	19
4.1.7. File pdf_parser.py di folder src/core/.....	20
4.1.8. File search.py di folder src/core/.....	21
4.1.9. File summary.py di folder src/core/.....	21
4.1.10. File database.py di folder src/db/.....	22
4.1.11. File models.py di folder src/db/.....	22
4.1.12. File keyword_input.py di folder src/ui/components/.....	23
4.1.13. File result_card.py di folder src/ui/components/.....	23
4.1.14. File main_window.py di folder src/ui/.....	24
4.1.15. File search_page.py di folder src/ui/.....	24
4.1.16. File summary_page.py di folder src/ui/.....	25
4.1.17. File file_utils.py di folder src/utils/.....	25
4.1.18. File keyword_utils.py di folder src/utils/.....	25
4.1.19. File timer.py di folder src/utils/.....	26
4.1.20. File main.py di folder src/.....	26
4.2. Tata Cara Penggunaan Program.....	27
4.3. Hasil Pengujian.....	28
4.4. Analisis Hasil Pengujian.....	37
<b>BAB V KESIMPULAN, SARAN, DAN REFLEKSI.....</b>	<b>39</b>

5.1. Kesimpulan.....	39
5.2. Saran.....	39
5.3. Refleksi.....	39
<b>DAFTAR PUSTAKA.....</b>	<b>40</b>
<b>LAMPIRAN.....</b>	<b>40</b>

## BAB I

### DESKRIPSI TUGAS



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

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

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

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

Di dalam Tugas Besar 3 ini, Anda diminta untuk mengimplementasikan sistem yang dapat melakukan deteksi informasi pelamar berbasis dokumen CV digital. Metode yang akan digunakan untuk melakukan deteksi pola dalam CV adalah Boyer-Moore dan Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas kandidat melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali profil pelamar secara lengkap hanya dengan menggunakan CV digital.

## BAB II

### LANDASAN TEORI

#### 2.1. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) mencari pola (*pattern*) dalam teks secara berurutan dari kiri ke kanan (left-to-right), sama seperti algoritma brute force. Namun, algoritma ini menggeser pola dengan cara yang lebih cerdas dibandingkan dengan algoritma brute force.

KMP menggunakan informasi yang diperoleh dari proses pencocokan sebelumnya untuk menghindari pemeriksaan ulang karakter yang sudah pasti cocok, sehingga dapat meningkatkan efisiensi pencarian pola dalam teks. Algoritma ini membangun sebuah tabel yang disebut *border function* (fungsi pinggiran) yang menyimpan panjang kemiripan terpanjang antara awalan (prefiks) dan akhiran (sufiks) pola pada setiap posisi. Dengan menggunakan tabel ini, ketika terjadi ketidakcocokan karakter, pola dapat digeser ke posisi yang tepat tanpa harus kembali memeriksa karakter yang sudah cocok sebelumnya. Hal ini menjadikan algoritma KMP memiliki kompleksitas waktu pencarian yang lebih baik, yaitu  $O(m+n)$ , di mana  $O(m)$  adalah kompleksitas untuk menghitung fungsi pinggiran dan  $n$  adalah panjang teks, sehingga lebih efisien daripada algoritma brute force yang memiliki kompleksitas  $O(n \times m)$ , dengan  $m$  adalah panjang pola.

#### 2.2. Algoritma Boyer Moore (BM)

Algoritma Boyer-Moore (BM) adalah salah satu algoritma pencarian string (string matching) yang terkenal karena efisiensinya. Dikembangkan oleh R.M. Boyer dan J.S. Moore pada tahun 1977, algoritma ini dikenal memiliki kinerja yang sangat baik dalam praktik. Ide utamanya adalah melakukan perbandingan karakter dari posisi paling kanan pola (string yang dicari) ke kiri. Jika terjadi ketidakcocokan, algoritma ini menggunakan dua heuristik penting: aturan *bad-character shift* dan aturan *good-suffix shift*. Aturan *bad-character shift* memungkinkan pola digeser berdasarkan karakter yang tidak cocok di teks, sementara aturan *good-suffix shift* memanfaatkan bagian pola yang sudah cocok untuk menentukan pergeseran optimal. Dengan mekanisme pergeseran ini, Boyer-Moore seringkali dapat melompati banyak karakter dalam teks, sehingga mengurangi jumlah perbandingan yang diperlukan.

Kompleksitas algoritma terdiri dari tiga kasus yaitu kasus terbaik, kasus rata-rata, dan kasus terburuk. Kasus terbaik adalah  $O(N/M)$ , dengan  $N$  adalah panjang teks dan  $M$  adalah panjang pola. Efisiensi ini dicapai karena algoritma dapat membuat lompatan besar. Kasus rata-rata adalah  $O(N+M)$ . Kinerja rata-rata Boyer-Moore mendekati linear. Kasus terburuk adalah  $O(N \times M)$ . Meskipun jarang terjadi dalam praktik, kasus terburuk ini dapat muncul pada skenario tertentu.

### **2.3. Algoritma Aho-Corasick**

Algoritma Aho-Corasick adalah algoritma pencarian string yang dirancang untuk menemukan sekumpulan pola (kamus) secara bersamaan dalam sebuah teks masukan. Dikembangkan oleh Alfred V. Aho dan Margaret J. Corasick pada tahun 1975, algoritma ini efisien dalam mengidentifikasi semua kemunculan pola-pola dari kamus yang telah ditentukan di dalam teks. Cara kerjanya melibatkan dua tahap utama: pertama, membangun sebuah automata *finite state* dari semua pola dalam kamus. Automata ini biasanya direpresentasikan sebagai *trie* (prefix tree) yang diperkaya dengan "tautan kegagalan" (*failure links*). Tautan kegagalan ini memungkinkan transisi yang cepat antar node di automata ketika terjadi ketidakcocokan, sehingga algoritma dapat melanjutkan pencarian tanpa harus memulai dari awal. Kedua, melakukan pencarian di dalam teks menggunakan automata yang telah dibangun.

Kompleksitas algoritma terdiri dari dua fase yaitu fase pra-pemrosesan dan fase pencarian. Fase pra-pemrosesan (Membangun Automata):  $O(L)$ , dengan  $L$  adalah total panjang semua pola dalam kamus. Ini adalah biaya satu kali. Fase encarian (Mencocokkan Teks):  $O(N+K)$ , dengan  $N$  adalah panjang teks dan  $K$  adalah jumlah total kemunculan pola yang ditemukan. Secara keseluruhan, sering disebut  $O(N+L)$ .

### **2.4. Levenshtein Distance**

*Levenshtein Distance*, juga dikenal sebagai *Edit Distance*, adalah metrik untuk mengukur tingkat perbedaan antara dua urutan karakter (string). Konsep ini diperkenalkan oleh matematikawan Soviet Vladimir Levenshtein pada tahun 1965. Jarak Levenshtein didefinisikan sebagai jumlah minimum operasi suntingan karakter tunggal yang diperlukan untuk mengubah satu string menjadi string lain. Operasi suntingan yang diperbolehkan meliputi: penyisipan (menambahkan karakter), penghapusan (menghapus karakter), atau penggantian (mengubah satu karakter menjadi karakter lain). Semakin kecil nilai *Levenshtein Distance*, semakin mirip kedua string tersebut. Algoritma ini banyak digunakan dalam aplikasi seperti koreksi ejaan, biometrik, dan analisis plagiarisme. Perhitungannya umumnya dilakukan menggunakan pendekatan pemrograman dinamis, dengan mengisi matriks yang merepresentasikan jarak antara semua prefiks dari kedua string.

Kompleksitas algoritma terdiri dari dua jenis yaitu kompleksitas waktu dan kompleksitas ruang. Kompleksitas waktu adalah  $O(M \times N)$ , dengan  $M$  dan  $N$  adalah panjang dari kedua string yang dibandingkan. Ini karena algoritma mengisi matriks berukuran  $(M+1) \times (N+1)$ . Kompleksitas ruang adalah  $O(M \times N)$  untuk menyimpan matriks. Namun, dapat dioptimalkan menjadi  $O(\min(M,N))$  jika hanya dua baris matriks yang disimpan.

## **2.5. Sistem ATS (Applicant Tracking System)**

Applicant Tracking System (ATS) adalah perangkat lunak yang dirancang untuk mengotomatisasi dan menyederhanakan proses rekrutmen dan perekrutan. Sistem ini membantu perusahaan mengelola, melacak, dan menyaring lamaran kerja secara efisien dari sejumlah besar kandidat. Fungsi utama ATS mencakup pengumpulan lamaran secara online, penyimpanan database kandidat, penyaringan awal berdasarkan kata kunci, penjadwalan wawancara, dan manajemen komunikasi dengan pelamar. ATS sangat penting bagi perusahaan modern karena kemampuannya untuk mengotomatisasi tugas-tugas administratif, mempercepat proses seleksi, dan membantu mengidentifikasi kandidat yang paling sesuai dengan cepat melalui analisis kata kunci pada CV.

## **2.6. Curriculum Vitae (CV)**

Curriculum Vitae (CV) adalah dokumen ringkasan yang digunakan oleh pencari kerja untuk menyajikan kualifikasi, pengalaman profesional, keterampilan, dan riwayat pendidikan mereka kepada calon pemberi kerja. Tujuan utama CV adalah untuk memberikan gambaran komprehensif tentang latar belakang kandidat dan meyakinkan manajer perekrutan bahwa mereka adalah pilihan yang tepat untuk posisi yang dilamar, sehingga layak untuk diundang wawancara. CV umumnya mencakup informasi kontak, ringkasan profil atau objektif karier, daftar pengalaman kerja, riwayat pendidikan, serta daftar keterampilan (teknis dan *soft skills*). Dalam era digital, format dan pemilihan kata kunci dalam CV menjadi sangat penting agar dapat dikenali dan lolos saringan awal oleh sistem ATS.

## **2.7. Penjelasan Mengenai Aplikasi Web yang Dibangun**

Aplikasi ini adalah Sistem Pelacakan Pelamar (ATS) berbasis CV digital yang dikembangkan menggunakan Python. Fokus utamanya adalah membantu perusahaan menyaring kandidat secara efisien dengan mengekstraksi dan mencocokkan informasi dari CV digital (PDF). Sistem ini memanfaatkan algoritma Pattern Matching seperti Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick untuk pencarian kata kunci cepat (exact match), serta Levenshtein Distance untuk pencarian yang lebih fleksibel (fuzzy match) guna menangani kesalahan ketik. Seluruh data kandidat, termasuk detail profil dan lokasi CV, disimpan dalam basis data MySQL. Aplikasi ini juga memiliki antarmuka pengguna intuitif untuk HR atau perekrut, yang memungkinkan mereka mencari, melihat ringkasan, dan mengakses CV asli kandidat dengan mudah.

## **BAB III**

### **ANALISIS PEMECAHAN MASALAH**

#### **3.1. Langkah Pemecahan Masalah**

Berikut beberapa langkah dalam memecahkan masalah :

- Enkripsi Data: Seluruh data profil pelamar kerja yang sensitif akan dienkripsi sebelum disimpan dalam database untuk memastikan kerahasiaannya. Enkripsi ini dilakukan menggunakan algoritma RSA, di mana setiap kolom yang berisi data sensitif akan dienkripsi dengan kunci publik dan disimpan dalam bentuk heksadesimal. Adapun dalam implementasi nyatanya, enkripsi dilakukan setelah memasukkan data mentah ke basis data yang diberikan oleh asisten. Hal itu dilakukan karena *seeding* basis data yang diberikan masih dalam data asli yang belum dienkripsi, sehingga mengenkripsi data (mengedit tabel) setelah dimasukkan lebih efektif dalam pengerjaannya.
- Ekstraksi Data : Data CV dalam format pdf akan diekstraksi menjadi sebuah teks string yang sumber *path*-nya berasal dari basis agar bisa diproses menggunakan algoritma pencarian.
- Pencarian Data: Program pertama kali akan melakukan pencocokan dengan metode *exact matching* yang terdiri dari tiga algoritma yaitu KMP, Boyer-Moore, dan Aho Corasick. Pengguna akan memilih dari ketiga algoritma tersebut untuk melakukan *searching*. Jika untuk masing-masing *keyword* dan masing-masing CV tidak ditemukan kecocokan, maka dilakukan pencarian dengan algoritma *fuzzy matching* yang memanfaatkan algoritma Levenshtein Distance untuk menangani kesalahan ketik atau variasi kata. Setelah data yang diperoleh muncul, pengguna dapat melihat CV asli atau *summary* dari CV tersebut. Summary sendiri akan menggunakan algoritma Regular Expression untuk mencari kumpulan data spesifik seperti *skills*, *education*, dan lainnya.
- Deskripsi Data: Setelah pencarian dilakukan dan data ditemukan, sistem perlu mendekripsi data yang telah terenkripsi sebelum menampilkannya kepada pengguna.

#### **3.2. Proses Pemetaan Masalah**

Berikut merupakan pemetaan masalah untuk masing-masing algoritma.

##### **3.2.1. Algoritma Knuth-Morris-Pratt (KMP)**

Algoritma KMP bekerja dengan dua fase utama:

- Pembentukan LPS (Longest Prefix Suffix) Array: Di fase ini, KMP menghitung array yang menyimpan panjang awalan terpanjang dari pola yang juga merupakan akhiran dari pola tersebut. Array LPS ini memungkinkan algoritma untuk melompat dan

melanjutkan pencarian dengan menghindari perbandingan berulang pada posisi yang sudah dipastikan tidak cocok.

- Pencocokan Teks dengan Pola: Algoritma menggunakan array LPS yang sudah dibangun untuk mempercepat proses pencocokan. Jika karakter dalam teks cocok dengan pola, maka pencocokan dilanjutkan. Jika tidak cocok, nilai dalam array LPS digunakan untuk menentukan seberapa banyak pola harus digeser tanpa perlu memeriksa karakter yang sudah diketahui tidak akan cocok.

Keunggulan algoritma KMP mengurangi jumlah perbandingan karakter yang tidak perlu dengan memanfaatkan array LPS. Ini menghindari pencocokan yang berulang dan meningkatkan kecepatan pencarian. Dengan menggunakan LPS, KMP dapat memproses teks dengan efisien, menghindari pemrosesan ulang pada bagian pola yang sudah diketahui.

Kelemahan algoritma KMP memerlukan waktu untuk membangun array LPS, meskipun proses ini efisien, waktu persiapan ini tetap diperlukan sebelum pencocokan dapat dimulai.

### 3.2.2. Algoritma Boyer-Moore (BM)

Seperti halnya KMP, algoritma Boyer-Moore juga bekerja dengan dua fase utama :

- Fase pertama dalam algoritma Boyer-Moore adalah membangun tabel Bad Character Rule, yang menyimpan posisi terakhir setiap karakter dalam pola. Tabel ini sangat berguna untuk menentukan seberapa banyak pola dapat digeser saat terjadi ketidakcocokan antara pola dan teks. Tabel ini dihitung hanya sekali di awal dan memiliki waktu komputasi  $O(m)$ , di mana  $m$  adalah panjang pola. Ini memastikan bahwa pencocokan dapat dilakukan dengan lebih cepat saat proses pencarian dimulai.
- Tahap selanjutnya adalah pencocokan. Algoritma Boyer-Moore mulai mencocokkan pola dengan teks dari belakang ke depan. Dimulai dari akhir pola, algoritma memeriksa apakah karakter dalam pola cocok dengan karakter pada posisi yang sesuai dalam teks. Jika terdapat kecocokan, algoritma bergerak ke karakter berikutnya dalam pola dan teks. Namun, jika terjadi ketidakcocokan, algoritma menggunakan informasi dari tabel Bad Character untuk memutuskan berapa banyak posisi pola yang dapat digeser. Setelah terjadi ketidakcocokan antara pola dan teks, pola akan digeser menggunakan informasi yang terdapat pada tabel Bad Character. Pergeseran ini memungkinkan pola untuk disesuaikan dengan karakter teks yang relevan, menghindari perbandingan lebih lanjut pada karakter yang telah diproses. Dengan pergeseran pola yang besar, algoritma Boyer-Moore dapat menghindari pemeriksaan karakter yang tidak relevan, meningkatkan efisiensi pencarian.

Kelebihan algoritma Boyer-Moore adalah efisiensi waktu pada kasus terbaik dan Rata-Rata. Algoritma Boyer-Moore sangat efisien dalam pencarian pola, terutama pada teks yang panjang. Pada kasus terbaik dan rata-rata, algoritma ini sering kali lebih cepat dibandingkan algoritma pencarian pola lainnya, seperti KMP dan brute-force. Hal ini karena kemampuannya untuk melakukan pergeseran pola yang besar (teks yang diberikan berupa alphabet), yang mengurangi jumlah perbandingan karakter yang harus dilakukan, khususnya ketika pola jarang muncul dalam teks.

Kekurangan Algoritma Boyer-Moore :

- Implementasi yang Lebih Kompleks, algoritma Boyer-Moore memiliki struktur yang lebih kompleks dibandingkan dengan algoritma lain seperti KMP atau brute-force. Penggunaan tabel Bad Character dan pergeseran pola yang tidak langsung menambah kompleksitas implementasi. Dibandingkan dengan algoritma pencarian yang lebih sederhana, Boyer-Moore memerlukan lebih banyak kode untuk menangani pembuatan dan pemrosesan tabel serta pencocokan karakter dari belakang ke depan.
- Menggunakan Lebih Banyak Memori, Boyer-Moore membutuhkan lebih banyak memori dibandingkan dengan algoritma pencarian pola lainnya karena penggunaan tabel Bad Character yang menyimpan posisi terakhir kemunculan setiap karakter dalam pola. Hal ini bisa menjadi masalah pada aplikasi yang memerlukan penggunaan memori rendah, terutama jika pola yang dicari memiliki banyak karakter yang berbeda.

Tidak Optimal untuk Semua Kasus:

Boyer-Moore tidak selalu menjadi pilihan terbaik untuk setiap jenis data atau pola. Jika pola sangat sering muncul dalam teks atau teks sangat mirip dengan pola, algoritma ini bisa jadi lebih lambat daripada alternatif lain seperti KMP, yang lebih konsisten dalam waktu eksekusinya terlepas dari kondisi teks atau pola.

### 3.2.3. Algoritma Aho-Corasick

Ada tiga fase utama dalam algoritma ini :

- Langkah pertama dalam algoritma Aho-Corasick adalah menambahkan pola-pola yang akan dicari ke dalam struktur data Trie. Trie adalah pohon pencarian yang menyimpan urutan karakter pola secara hierarkis. Setiap node dalam Trie mewakili karakter dalam pola, dan setiap jalur dari root ke node yang terletak di level akhir merepresentasikan pola itu sendiri. Fungsi `add_pattern()` digunakan untuk menambahkan pola satu per satu ke dalam Trie, serta memperbarui node, fungsi `fail`, dan fungsi output yang terkait dengan pola tersebut.
- Setelah pola-pola ditambahkan ke Trie, langkah berikutnya adalah membangun fungsi `fail`. Fungsi `fail` digunakan untuk mengatur link antara node-node yang tidak memiliki transisi langsung berdasarkan karakter yang ada pada input teks. Jika pencocokan karakter gagal, algoritma menggunakan `fail` link untuk kembali ke node yang relevan dan melanjutkan pencocokan tanpa memulai ulang dari awal. Fungsi

`build_failure_function()` melakukan hal ini dengan melakukan traversal menggunakan algoritma Breadth-First Search (BFS) dan menambahkan fail link untuk setiap node di Trie.:

- Setelah Trie dan fail function dibangun, pencocokan pola dengan teks dapat dimulai. Proses ini dimulai dari root node Trie dan mengikuti jalur berdasarkan karakter yang ditemukan di teks. Fungsi `get_next_state()` digunakan untuk mengembalikan state berikutnya berdasarkan input karakter teks. Setiap kali terjadi kecocokan atau saat fail link digunakan, output dari node terkait akan diupdate dan disimpan dalam result untuk mencatat kemunculan pola yang ditemukan. Fungsi `search()` menangani pencarian teks berdasarkan struktur Trie dan fail link yang telah dibangun.

Kelebihan utama dari algoritma Aho-Corasick adalah kemampuannya untuk melakukan pencarian terhadap banyak pola secara simultan dalam satu kali traversal teks. Hal ini sangat efisien jika dibandingkan dengan pencarian pola satu per satu menggunakan algoritma lain seperti KMP atau Boyer-Moore, yang membutuhkan perbandingan terpisah untuk setiap pola. Aho-Corasick sangat cepat saat mencari beberapa pola di dalam teks. Setelah preprocessing Trie dan fungsi fail selesai dibangun, pencarian hanya memerlukan satu kali traversal teks, sehingga algoritma ini sangat efisien untuk pencarian dalam teks besar dengan banyak pola yang harus dicocokkan.

Kekurangan Algoritma Aho-Corasick :

- Memori yang digunakan lebih besar. Aho-Corasick membutuhkan lebih banyak memori dibandingkan dengan algoritma lain seperti KMP atau Boyer-Moore. Hal ini disebabkan oleh kebutuhan untuk menyimpan Trie dan fail link untuk setiap karakter dalam pola. Jumlah memori yang diperlukan akan meningkat secara signifikan ketika ada banyak pola yang harus dicari, yang mungkin menjadi masalah dalam sistem dengan sumber daya terbatas.
- Overhead dalam preprocessing. Proses pembangunan Trie dan fungsi fail memerlukan waktu preprocessing yang dapat memakan waktu. Meskipun pencarian pola setelah preprocessing sangat cepat, waktu yang dibutuhkan untuk membangun struktur data awal dapat menjadi besar ketika jumlah pola yang dicari sangat banyak.
- Kurang Efisien untuk pencarian satu pola. Jika hanya ada satu pola yang akan dicari dalam teks, Aho-Corasick mungkin tidak lebih efisien daripada algoritma pencarian pola lainnya seperti KMP atau Boyer-Moore. Keuntungan dari Aho-Corasick lebih terlihat ketika banyak pola yang perlu dicocokkan secara bersamaan.

### 3.2.4. Algoritma Fuzzy Matching

Ada tiga fase utama dalam algoritma ini :

- Perhitungan Jarak Levenshtein (`levenshtein_distance`). Fungsi ini menghitung jarak Levenshtein antara dua string dengan menggunakan algoritma Dynamic Programming. Matriks dua dimensi dibangun untuk menghitung jumlah operasi yang diperlukan untuk mengubah satu string menjadi string lainnya. Setiap posisi dalam

matriks mewakili operasi (insert, delete, atau substitute) yang diperlukan pada indeks tertentu.

- Menghitung Rasio Kemiripan (levenshtein\_ratio). Setelah mendapatkan jarak Levenshtein antara dua string, rasio kemiripan dihitung dengan cara mencari jarak maksimum string dibandingkan dengan jarak Levenshtein lalu nilai itu dikurangkan dengan satu. Jika jarak Levenshtein lebih kecil, rasio kemiripan akan lebih tinggi, menunjukkan kesamaan yang lebih besar.
- Fungsi fuzzy\_search memecah teks menjadi kata-kata individu dan menghitung rasio kemiripan setiap kata dengan kata kunci menggunakan Levenshtein Distance. Jika rasio kemiripan untuk sebuah kata lebih tinggi dari ambang batas yang ditentukan (dalam hal ini 80% kemiripan), kata tersebut dianggap cocok, meskipun terdapat perbedaan ketik atau variasi minor dalam kata kunci.

Kelebihan dari Algoritma ini sangat berguna untuk mencari kata-kata yang mungkin memiliki kesalahan ketik, variasi kecil, atau perbedaan format, karena menggunakan Levenshtein Distance untuk menilai kemiripan antar kata.

Kekurangan Algoritma ini tidak dioptimalkan untuk mencari dalam daftar kata besar atau dokumen panjang. Dalam konteks aplikasi yang mengharuskan pencarian dalam teks besar (seperti pencarian teks di seluruh database), ini bisa menjadi lambat dan kurang efisien.

Sedikit catatan, semua algoritma tidak menghitung kemunculan yang saling *overlapping*.

### 3.3. Fitur Fungsional Aplikasi Web yang Dibangun

#### 3.3.1. Fitur

#### 3.3.2. Database ats\_db

Fitur database ini disediakan di aplikasi web yang dibangun untuk menyimpan informasi mengenai file cv dalam format PDF yang diambil dari folder data. Database bernama ats\_db ini memerlukan beberapa dependensi selain program Python dan MySQL yaitu secara manual menginstall berikut ini.

```
pip install faker sqlalchemy mysql-connector-python python-dotenv
```

Setelah itu, menambah file .env di direktori projek menggunakan template di README.md dengan mengganti your\_password menggunakan password dari username root yang dipakai di MySQL lalu mengubah portnya menyesuaikan dengan port MySQL yang dipakai jika tidak menggunakan port default (3306) dan dapat dilakukan seeding atau memasukkan nilai agar tidak null di setiap atribut yang awalnya null dan nantinya database pada MySQL di perangkat akan diperbarui yaitu dibuat database ats\_db baru dengan menjalankan ini di terminal.

```
python -m src.db.database
```

Lalu, database `ats_db` dapat ditampilkan dalam bentuk tabel di Visual Studio Code dengan cara menambah ekstensi SQLTools dan SQLTools MySQL/MariaDB/TiDB Driver yang keduanya disediakan oleh Matheus Teixeira. Setelah menambah ekstensi tersebut, membuka ekstensi SQLTools dengan memilih MySQL di Add New Connection. Pengisian untuk koneksi adalah ini connection namanya bebas, port sesuaikan dengan port yang dipakai di MySQL, di Database isi dengan '`ats_db`', pada baris Username isi dengan '`root`', pilih 'Save as plaintext in settings', di bagian Password mode isi password sesuai dengan password di MySQL, lalu klik 'Save Connection', dan klik 'Connect now'. Pada file sql yang dibuat otomatis setelah terjadi koneksi yang dilakukan maka isi dengan ini.

```
SELECT * FROM applicationdetail;  
SELECT * FROM applicantprofile;
```

Pada baris-baris *query* di atas pilih salah satu baris dengan cara diblok lalu klik kanan dan mengklik 'Run Selected Query' untuk melihat datanya dalam tampilan tabel.

### 3.4. Ilustrasi Kasus

Seorang HR Recruiter di sebuah perusahaan teknologi yang sedang mencari Software Engineer baru. Dia memiliki tumpukan ratusan CV dalam format PDF dari berbagai pelamar. Menyaringnya satu per satu secara manual akan memakan waktu berhari-hari, bahkan berminggu-minggu.

Inilah saatnya Sistem ATS berperan:

1. Ekstraksi Otomatis: Semua CV PDF dari pelamar secara otomatis diunggah dan teksnya diekstrak oleh sistem. Informasi penting seperti nama, kontak, riwayat pendidikan, pengalaman kerja, dan daftar skill langsung diproses dan disimpan ke database.
2. Pencarian Cepat: Dia tahu bahwa Dia membutuhkan insinyur dengan pengalaman di Python, React, dan SQL. Dia membuka aplikasi ATS Anda.
  - o Di kolom input kata kunci, Dia ketik: python, react, sql.
  - o Dia bisa memilih algoritma pencarian: misalnya, Dia mencoba Boyer-Moore (BM) karena Anda tahu ini cepat untuk pencocokan persis.
  - o Dia mengatur "Top Matches" ke 10 untuk melihat 10 kandidat terbaik.
  - o Dia klik tombol "Search".
3. Hasil Instan: Dalam hitungan milidetik, sistem menampilkan 10 CV teratas yang paling relevan.
  - o Dia dapat melihat daftar kandidat dengan informasi seperti: Nama Kandidat, Jumlah Kecocokan (misal: "3 kata kunci cocok"), dan daftar skill yang cocok beserta frekuensinya (misal: Python (5), React (3), SQL (2)).

- Di bagian bawah, sistem menampilkan: "Exact Match: 500 CVs scanned in 50ms. Fuzzy Match: 0 CVs scanned." (Karena semua kata kunci ditemukan secara persis).
4. Fuzzy Match untuk Presisi: Kemudian, Dia teringat bahwa ada juga skill PostgreSQL yang penting, tapi Dia lupa menuliskannya dengan benar dan hanya mengetik Postgre.
- Dia tambahkan Postgre ke daftar kata kunci dan klik "Search" lagi.
  - Kali ini, sistem mungkin menemukan beberapa kecocokan persis. Namun, untuk kata kunci Postgre yang mungkin tidak ditemukan persis di CV (karena tertulis PostgreSQL), sistem akan otomatis melakukan Fuzzy Matching menggunakan Levenshtein Distance.
  - Sistem akan menampilkan CV yang memiliki PostgreSQL meskipun Dia mengetik Postgre, karena jarak Levenshtein-nya sangat kecil.
  - Waktu eksekusi sekarang mungkin menunjukkan: "Exact Match: 500 CVs scanned in 55ms. Fuzzy Match: 100 CVs scanned in 120ms." (Menunjukkan waktu tambahan untuk pencarian fuzzy).
5. Melihat Detail & CV Asli:
- Dia tertarik pada satu kandidat bernama "Budi Santoso". Dia klik tombol "Summary" pada kartu CV-nya.
  - Sebuah halaman baru muncul, menampilkan ringkasan Budi: Nama, Kontak, ringkasan profil, daftar keahlian, pengalaman kerja lengkap (jabatan, tanggal), dan riwayat pendidikan yang diekstraksi dari CV-nya.
  - Untuk memverifikasi informasi atau melihat detail lain yang mungkin terlewat, Anda mengklik tombol "View CV", dan file PDF CV asli Budi pun terbuka.

Dengan sistem ini, Dia dapat menyaring ribuan CV dalam hitungan menit, bukan berhari-hari, memastikan Dia tidak melewatkkan kandidat potensial hanya karena beban kerja manual yang masif.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1. Spesifikasi Teknis Program

##### 4.1.1. Struktur Program ATS

```
.  
|   └── MySQLCV.session.sql # Skrip sesi database MySQL  
|   └── README.md          # Dokumentasi utama proyek  
|   └── data/              # Berisi kumpulan data CV (berkas PDF) yang dikategorikan berdasarkan profesi  
|       |   └── ACCOUNTANT/  
|       |   └── ADVOCATE/  
|       |   └── AGRICULTURE/  
|       |   ... (dan kategori profesi lainnya)  
|   └── doc/               # Dokumentasi tambahan atau aset proyek  
|       └── creators.jpg  
└── src/                # Kode sumber aplikasi utama  
    └── core/              # Implementasi inti algoritma pencarian dan pemrosesan teks  
        |   └── aho_corasick.py # Algoritma Aho-Corasick untuk pencarian string  
        |   └── boyer_moore.py # Algoritma Boyer-Moore untuk pencarian string  
        |   └── encryption.py  # Logika enkripsi (mungkin untuk data atau kredensial)  
        |   └── kmp.py         # Algoritma Knuth-Morris-Pratt (KMP) untuk pencarian string  
        |   └── levenshtein.py # Algoritma Levenshtein untuk perhitungan jarak edit (kesamaan string)  
        |   └── pdf_parser.py # Modul untuk mengekstrak teks dari berkas PDF  
        |   └── search.py      # Logika utama untuk melakukan pencarian CV  
        |       └── summary.py # Logika untuk membuat ringkasan CV  
    └── db/                 # Modul untuk interaksi dengan database  
        |   └── ats.sql        # Skema database untuk Applicant Tracking System  
        |   └── database.py   # Koneksi dan operasi database  
        |   └── encryption.py # (Kemungkinan) Enkripsi terkait database  
        |   └── models.py     # Definisi model data untuk ORM/mapping database  
        |       └── tubes3_seeding.sql# Skrip untuk mengisi database dengan data awal (seeding)  
    └── main.py            # Titik masuk utama aplikasi (main entry point)  
    └── ui/                # Antarmuka Pengguna (User Interface) aplikasi  
        |   └── components/   # Komponen UI yang dapat digunakan kembali  
            |       └── keyword_input.py # Komponen input kata kunci
```

```

    |   |   └── result_card.py # Komponen untuk menampilkan hasil pencarian
    |   ├── main_window.py # Jendela utama aplikasi
    |   ├── search_page.py # Halaman/view untuk fungsi pencarian CV
    |   └── summary_page.py # Halaman/view untuk menampilkan ringkasan CV
    └── utils/      # Utility functions (fungsi pembantu umum)
        ├── file_utils.py  # Utilitas terkait operasi berkas
        ├── keyword_utils.py # Utilitas terkait pemrosesan kata kunci
        └── timer.py       # Utilitas untuk pengukuran waktu (mis. kinerja pencarian)

```

#### 4.1.2. File aho\_corasick.py di folder src/core/

- **Metode**

Metode	Deskripsi
def add_pattern(pattern, trie, fail, output)	Fungsi ini menambahkan pola ke dalam struktur trie, memperbarui node trie, serta mengaitkan pola yang ditemukan dengan node akhir dalam struktur output.
build_failure_function(trie, fail, output)	Fungsi ini membangun fungsi kegagalan untuk trie menggunakan pendekatan BFS, memastikan bahwa setiap node memiliki tautan kegagalan yang mengarah ke node yang benar jika transisi tidak ditemukan.
get_next_state(current_state, char, trie, fail)	Fungsi ini menggabungkan pencarian dalam trie dengan fungsi kegagalan untuk mencari state berikutnya berdasarkan karakter yang diberikan.
search(text, trie, fail, output)	Fungsi ini mencari pola dalam teks menggunakan trie dan fungsi kegagalan, mengembalikan hasil pencocokan pola dalam teks.
aho_corasick_search(keywords, normalized_cv_content)	Fungsi utama untuk melakukan pencarian pola menggunakan algoritma Aho-Corasick, yang pertama-tama menambahkan pola ke dalam trie, membangun fungsi kegagalan, dan akhirnya mencari pola dalam teks yang telah dinormalisasi.

normalize_text(text)	Fungsi ini menormalisasi teks dengan mengonversinya menjadi huruf kecil dan menghapus karakter non-alfanumerik, serta mengganti spasi berlebih dengan satu spasi.
----------------------	---

#### 4.1.3. File boyer\_moore.py di folder src/core/

- **Metode**

Metode	Deskripsi
def boyer_moore_search(text: str, pattern: str) -> int:	Menghitung kemunculan pattern string dengan text string yang diberikan. Kemunculan yang saling overlapping tidak akan terhitung double.

#### 4.1.4. File encryption.py di folder src/core/

- **Metode**

Metode	Deskripsi
def mod_inverse(e, phi_n)	Menghitung invers modular dari e terhadap phi_n menggunakan sympy.mod_inverse().
def string_to_int(message)	Mengubah setiap karakter dalam pesan menjadi angka berdasarkan kode ASCII-nya.
def int_to_string(int_list)	Mengubah setiap karakter dalam pesan menjadi angka berdasarkan kode ASCII-nya.
def int_to_hex(int_list)	Mengubah daftar angka menjadi representasi heksadesimal, setiap angka diwakili dalam format 4 digit
def hex_to_int(hex_string)	Mengonversi string heksadesimal menjadi daftar angka dengan memecah string dan mengonversi tiap bagian ke angka desimal.
def encrypt(message) -> str	Mengenkripsi pesan menggunakan RSA

	dengan mengonversi pesan menjadi angka, mengenkripsi setiap angka, dan mengonversinya ke dalam format heksadesimal.
def decrypt(ciphertext_hex) -> str	Mendekripsi ciphertext dalam format heksadesimal dengan mengubahnya kembali ke angka, mendekripsinya menggunakan kunci privat, dan mengonversinya kembali menjadi string

#### 4.1.5. File kmp.py di folder src/core/

- **Metode**

Metode	Deskripsi
compute_lps_array(pattern: str) -> List[int]	Fungsi ini menghitung array LPS (Longest Prefix Suffix) yang digunakan dalam algoritma KMP untuk mengoptimalkan pencocokan pola dengan teks.
kmp_search(text: str, pattern: str) -> List[int]	Fungsi ini mencari semua kemunculan pola dalam teks menggunakan algoritma KMP dan mengembalikan daftar indeks di mana pola tersebut ditemukan.

#### 4.1.6. File levenshtein.py di folder src/core/

- **Metode**

Metode	Deskripsi
levenshtein_distance(s1: str, s2: str) -> int	Fungsi ini menghitung jarak Levenshtein (jumlah operasi insert, delete, dan substitute) antara dua string untuk mengukur perbedaan mereka.
levenshtein_ratio(s1: str, s2: str) -> float	Fungsi ini menghitung rasio kemiripan antara dua string berdasarkan jarak Levenshtein, mengembalikan nilai antara 0 dan 1, di mana 1 berarti string sangat mirip.

fuzzy_search(text: str, keyword: str, threshold: float) -> int	Fungsi ini mencari dan menghitung jumlah kemunculan kata yang mirip dengan keyword dalam text berdasarkan rasio kemiripan Levenshtein yang lebih besar atau sama dengan ambang batas (threshold).
--	---

#### 4.1.7. File pdf\_parser.py di folder src/core/

- **Metode**

Metode	Deskripsi
extract_text_from_pdf_raw(pdf_path: str) -> str   None	Fungsi ini mengekstrak teks mentah dari file PDF menggunakan pdfplumber dan mengembalikannya dalam bentuk string, atau None jika terjadi kesalahan atau file tidak ditemukan.
combine_and_normalize_text(raw_extract_text: str) -> str	Fungsi ini membersihkan dan menggabungkan teks yang diekstraksi dengan menghapus baris kosong dan memformatnya menjadi teks yang lebih terstruktur, tanpa mengubah kapitalisasi.
_extract_phone_numbers(text: str) -> list[str]	Fungsi ini mencari dan mengekstrak nomor telepon dari teks menggunakan ekspresi reguler, dan mengembalikan daftar nomor telepon yang valid.
_extract_skills(text: str) -> list[str]	Fungsi ini mengekstrak keterampilan dari bagian "Skills" dalam teks, memecahnya berdasarkan tanda koma, spasi, atau bullet points, dan membersihkan entri kosong atau tidak relevan.
_extract_job_history(text: str) -> list[dict]	Fungsi ini mengekstrak riwayat pekerjaan dari teks, mencari setiap entri pekerjaan yang berisi jabatan, periode, nama perusahaan, dan deskripsi pekerjaan, kemudian mengembalikannya dalam bentuk daftar dictionary.

<code>_extract_education(text: str) -&gt; list[dict]</code>	Fungsi ini mengekstrak informasi pendidikan dari teks, mencari jurusan kuliah, institusi, dan periode waktu pendidikan, lalu mengembalikannya dalam bentuk daftar dictionary.
<code>parse_pdf_to_text(pdf_path: str = None, text_content: str = None) -&gt; str   None</code>	Fungsi ini mengekstrak dan menormalkan teks dari file PDF atau teks yang diberikan, mengembalikan teks yang telah diproses atau None jika gagal.
<code>parse_pdf_to_text_and_extract_info(pdf_path: str = None, text_content: str = None) -&gt; dict</code>	Fungsi ini mengekstrak teks mentah dari PDF atau teks yang diberikan, menormalkannya, dan kemudian mengekstrak informasi terstruktur seperti nomor telepon, keterampilan, riwayat pekerjaan, dan pendidikan, lalu mengembalikannya dalam bentuk dictionary.

#### 4.1.8. File search.py di folder src/core/

- **Metode**

Metode	Deskripsi
<code>normalize_text(text: str) -&gt; str</code>	Fungsi ini mengonversi teks menjadi huruf kecil dan menggantikan karakter non-alfanumerik dengan spasi, kemudian menghapus spasi berlebih.
<code>perform_search(keywords_tuple: Tuple[str, ...], selected_algorithm: str, top_n: int) -&gt; Tuple[List[Dict], int, Dict]</code>	Fungsi ini mencari kata kunci dalam file CV menggunakan algoritma pencarian yang dipilih (Aho-Corasick, KMP, Boyer-Moore, atau pencocokan Fuzzy), menghitung jumlah kecocokan, dan mengembalikan daftar hasil pencarian terbaik beserta waktu eksekusi untuk pencocokan tepat dan fuzzy.

#### 4.1.9. File summary.py di folder src/core/

- **Metode**

Metode	Deskripsi
<code>get_candidate_summary(applicant_id: int)</code>	Fungsi ini mengambil rangkuman data

full_cv_path: str, cv_content: str = None) -> dict	kandidat berdasarkan applicant_id yang diberikan, menggabungkan informasi dari database seperti profil pelamar dan detail aplikasi, serta menambahkan data yang diekstrak dari CV (keterampilan, riwayat pekerjaan, dan pendidikan) tanpa memodifikasi database.
---	--

#### 4.1.10. File database.py di folder src/db/

- **Metode**

Metode	Deskripsi
initialize_engine_and_session()	Fungsi ini menginisialisasi koneksi database menggunakan SQLAlchemy dengan mengatur engine dan session untuk aplikasi, dan memastikan koneksi ke database berhasil.
get_db_session()	Fungsi ini adalah context manager yang menyediakan sesi database yang aman untuk digunakan, memastikan bahwa sesi dibuka dan ditutup dengan benar, serta menangani rollback jika terjadi exception selama transaksi.
setup_database_from_sql(sql_file_path: str, drop_db_if_exists: bool = True)	Fungsi ini membuat database baru jika belum ada, menghapus database yang sudah ada jika diinginkan, dan mengeksekusi file SQL untuk setup dan seeding database, termasuk menangani error jika proses gagal.

#### 4.1.11. File models.py di folder src/db/

- **Class**

Metode	Deskripsi
ApplicantProfile	Kelas ini adalah model SQLAlchemy yang merepresentasikan tabel ApplicantProfile dalam database. Tabel ini menyimpan informasi tentang pelamar, seperti applicant_id, first_name, last_name, date_of_birth, address, dan phone_number.

ApplicationDetail	Kelas ini adalah model SQLAlchemy yang merepresentasikan tabel ApplicationDetail dalam database. Tabel ini menyimpan detail aplikasi pelamar, termasuk detail_id, applicant_id (yang merujuk ke ApplicantProfile), application_role, dan cv_path. Beberapa kolom yang berkaitan dengan caching telah dihapus dari model ini.
-------------------	--

#### 4.1.12. File keyword\_input.py di folder src/ui/components/

- **Class dan fungsi**

<b>Metode</b>	<b>Deskripsi</b>
KeywordInput	Kelas ini adalah widget berbasis QWidget yang memungkinkan pengguna untuk memasukkan kata kunci yang dipisahkan dengan koma. Widget ini mencakup:
keywords(self) -> list[str]	Fungsi ini mengembalikan daftar kata kunci yang dimasukkan oleh pengguna, memisahkan kata kunci berdasarkan koma dan membersihkan spasi di sekitar setiap kata kunci yang dimasukkan, serta mengabaikan kata kunci kosong.

#### 4.1.13. File result\_card.py di folder src/ui/components/

- **Metode**

<b>Metode</b>	<b>Deskripsi</b>
_build_ui()	Membangun UI untuk widget ResultCard dengan layout vertikal yang terdiri dari informasi kandidat, daftar kata kunci yang cocok, dan dua tombol untuk melihat ringkasan dan membuka CV.
open_cv_pdf()	Membuka file PDF CV yang sesuai dengan cv_path menggunakan aplikasi default yang terpasang di sistem.

4.1.14. File main\_window.py di folder src/ui/

- **Metode**

<b>Metode</b>	<b>Deskripsi</b>
_show_summary_page	Mengarahkan aplikasi ke halaman ringkasan dan memuat data kandidat berdasarkan applicant_id, cv_path, dan cv_content.
_show_search_page	Mengarahkan aplikasi kembali ke halaman pencarian.

4.1.15. File search\_page.py di folder src/ui/

- **Class**

<b>Metode</b>	<b>Deskripsi</b>
SearchWorker	Kelas pekerja yang diturunkan dari QObject, digunakan untuk menjalankan operasi pencarian dalam thread terpisah menggunakan fungsi perform_search. Kelas ini mengirimkan hasil pencarian, total jumlah CV yang dipindai, dan informasi waktu saat pencarian selesai. Jika terjadi kesalahan, ia akan mengirimkan pesan kesalahan.
AlgorithmToggle	Widget toggle kustom yang memungkinkan pengguna memilih antara algoritma pencarian yang berbeda (misalnya KMP, Boyer-Moore, Aho-Corasick) dengan animasi geser. Widget ini mengirimkan sinyal dengan nama algoritma yang dipilih saat ada pemilihan.
SearchPage	Halaman widget yang menyediakan antarmuka bagi perekut untuk mencari CV berdasarkan kata kunci. Halaman ini mencakup:

4.1.16. File summary\_page.py di folder src/ui/

- **Class**

Metode	Deskripsi
SummaryPage	halaman yang menampilkan informasi rinci tentang kandidat, termasuk informasi pribadi, keterampilan, riwayat pekerjaan, dan pendidikan, dengan tampilan yang responsif dan memungkinkan navigasi kembali ke halaman pencarian.

4.1.17. File file\_utils.py di folder src/utils/

- **Metode**

Metode	Deskripsi
read_file_content(file_path: str, encoding: str = 'utf-8') -> str   None	Fungsi ini membaca seluruh konten dari file teks yang diberikan berdasarkan path dan encoding yang ditentukan, mengembalikan konten file sebagai string atau None jika terjadi kesalahan.
write_to_file(file_path: str, content: str, encoding: str = 'utf-8', mode: str = 'w') -> bool	Fungsi ini menulis konten ke file yang ditentukan, membuat direktori jika belum ada, dan memungkinkan pemilihan mode penulisan (overwrite atau append), mengembalikan True jika berhasil dan False jika gagal.
check_path_exists(path: str) -> bool	Fungsi ini memeriksa apakah path (baik file atau folder) yang diberikan ada, mengembalikan True jika ada dan False jika tidak.
get_file_extension(file_path: str) -> str	Fungsi ini mengambil ekstensi file dari path yang diberikan, mengembalikan ekstensi file (misal: .pdf, .txt) atau string kosong jika tidak ada ekstensi.

4.1.18. File keyword\_utils.py di folder src/utils/

- **Metode**

<b>Metode</b>	<b>Deskripsi</b>
normalize_text(text: str) -> str	Fungsi ini menormalisasi teks dengan mengonversinya menjadi huruf kecil dan menghapus tanda baca, mengembalikan teks yang sudah diproses.
tokenize_text(text: str) -> list[str]	Fungsi ini melakukan tokenisasi pada teks untuk memisahkan teks menjadi daftar kata (token), dengan menggunakan regex untuk menangkap kata alfanumerik.
remove_stopwords(tokens: list[str], stopwords: set[str] = None) -> list[str]	Fungsi ini menghapus kata-kata yang dianggap tidak relevan (stop words) dari daftar token, dengan pilihan untuk menggunakan stop words default atau yang disediakan.
extract_unique_keywords(text: str, normalize: bool = True, remove_sw: bool = True) -> list[str]	Fungsi ini mengekstrak daftar kata kunci unik dari teks, dengan opsi untuk menormalkan teks dan menghapus stop words, mengembalikan daftar kata kunci yang sudah diproses dan diurutkan.

#### 4.1.19. File timer.py di folder src/utils/

- **Metode**

<b>Metode</b>	<b>Deskripsi</b>
start_timer() -> float	Fungsi ini memulai timer dan mengembalikan waktu awal (dalam detik) menggunakan time.perf_counter(), yang digunakan untuk mengukur durasi eksekusi suatu operasi.
stop_timer(start_time: float, operation_name: str = "Operation") -> float	Fungsi ini menghentikan timer, menghitung durasi sejak waktu yang diberikan oleh start_timer(), dan mencetak waktu eksekusi dalam milidetik, mengembalikan durasi eksekusi tersebut.

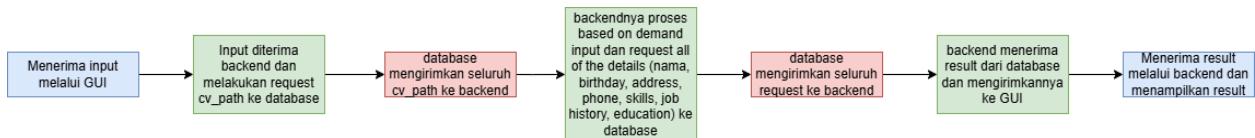
#### 4.1.20. File main.py di folder src/

- **Metode**

Metode	Deskripsi
main()	Fungsi ini adalah titik masuk aplikasi. Fungsi ini membuat objek QApplication untuk mengelola event loop Qt, kemudian membuat dan menampilkan jendela utama (MainWindow) dalam ukuran maksimal, dan akhirnya menjalankan aplikasi dengan app.exec().

## 4.2. Tata Cara Penggunaan Program

Berikut adalah cara penggunaan program setelah sudah menginstall program yang dibutuhkan.



1. Jika sudah melakukan instalasi Python, MySQL, dependensi lainnya, serta telah menambahkan .env yang disediakan di README.md, maka berikutnya dapat dilakukan seeding dari tubes3\_seeding.sql.

```
python -m src.db.database
```

2. Lalu, berikutnya karena telah mengimplementasikan bonus enkripsi data profil applicant, maka dapat menjalankan ini.

```
python -m src.db.encryption
```

3. Setelah itu, untuk menjalankan aplikasi utamanya adalah ini.

```
python -m src.main
```

4. [INPUT] Pengguna memasukkan sembarang kata kunci (*keyword*) yang ingin dicari banyaknya file CV dalam format PDF yang ada dan banyaknya kata kunci dipisahkan oleh koma seperti contoh yang telah disediakan pada tampilan aplikasi.
5. [INPUT] Pengguna berikutnya memilih salah satu dari antara 3 algoritma mana yang mau digunakan, yaitu KMP, Boyer-Moore, atau Aho-Corasick.
6. [INPUT] Pengguna dapat mengatur berapa banyaknya hasil pencarian yang diinginkan yaitu mengubah pada bagian ‘Top Matches:’ dengan jumlah default yang dibuat adalah 8 buah file CV dengan minimum 0 dan maksimal 600 buah file CV for.

7. [INPUT] Pengguna berikutnya menekan tombol ‘Search’ untuk memulai pencarian file cv dengan kata kunci dan algoritma yang diinginkan.
8. [OUTPUT] Aplikasi akan memberikan lama waktu pencarian dan lama *fuzzy match*, lalu pada kolom hasil pencarian ditunjukkan hasil file CV yang didapat dengan menunjukkan nama pemilik CV, kata kunci, banyaknya kata kunci yang didapat pada file cv tersebut, dan terdapat tombol untuk melihat ringkasan CV pada tombol ‘Summary’ dan melihat isi dari CV dengan tombol ‘View CV’.
9. [OUTPUT] Jika pengguna menekan tombol ‘Summary’ maka aplikasi akan menunjukkan ringkasan CV berupa nama pemilik CV, data pemilik (tanggal lahir, alamat, dan nomor telepon), keahlian yang dimiliki (*Skills:*), riwayat pekerjaan (*Job History:*), dan pendidikan yang telah ditempuh (*Education:*).
10. [OUTPUT] Setelah itu, pengguna dapat menekan ‘Back to Search’ untuk kembali ke menu awal yaitu tampilan pencarian yang dilakukan, lalu pengguna menekan tombol ‘View CV’
11. [OUTPUT] Aplikasi akan menunjukkan secara langsung isi dari CV yang terdapat pada folder data/<role>/ yang sejajar dengan folder src yang berisi program dari aplikasi ini. Aplikasi akan menunjukkan isi dari CV tersebut dengan menemukan jalan atau *cv\_path* dari hasil file CV yang ingin ditunjukkan isinya.

### 4.3. Hasil Pengujian

#### 4.3.1. Test Case 1

Input
name
Output

The screenshot shows the 'Applicant Tracking System - CV Search' interface. The search bar contains the keyword 'name'. The 'Select Algorithm' dropdown is set to 'Boyer-Moore'. The search results are displayed in a grid format:

- Ikhw4n ALH4KIM**: 22 matches. Matched keywords: 1. name: 22 occurrences. Buttons: Summary, View CV.
- Ari3L HerfR150n**: 20 matches. Matched keywords: 1. name: 20 occurrences. Buttons: Summary, View CV.
- ar13l h3rfr150n**: 14 matches. Matched keywords: 1. name: 14 occurrences. Buttons: Summary, View CV.
- ar13l h3rfr150n**: 14 matches. Matched keywords: 1. name: 14 occurrences. Buttons: Summary, View CV.
- H41k4l 455y4uq1**: 14 matches. Matched keywords: 1. name: 14 occurrences. Buttons: Summary, View CV.
- Ari3L HerfR150n**: 13 matches. Matched keywords: 1. name: 13 occurrences. Buttons: Summary, View CV.
- ALaNd MuLiA**: 12 matches. Matched keywords: 1. name: 12 occurrences. Buttons: Summary, View CV.
- 4l4nd MuL14**: 11 matches. Matched keywords: 1. name: 11 occurrences. Buttons: Summary, View CV.

Exact Search Time (Boyer-Moore): 272.92 ms | Fuzzy Match: 0.00 ms

Total CVs Processed: 600

#### 4.3.2. Test Case 2

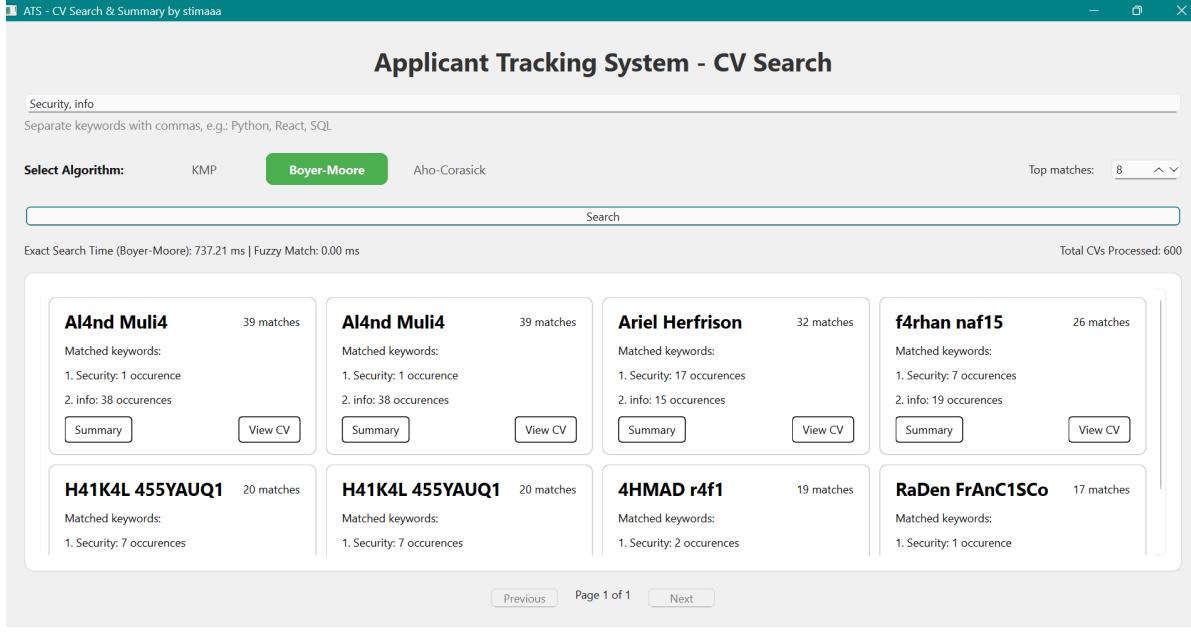
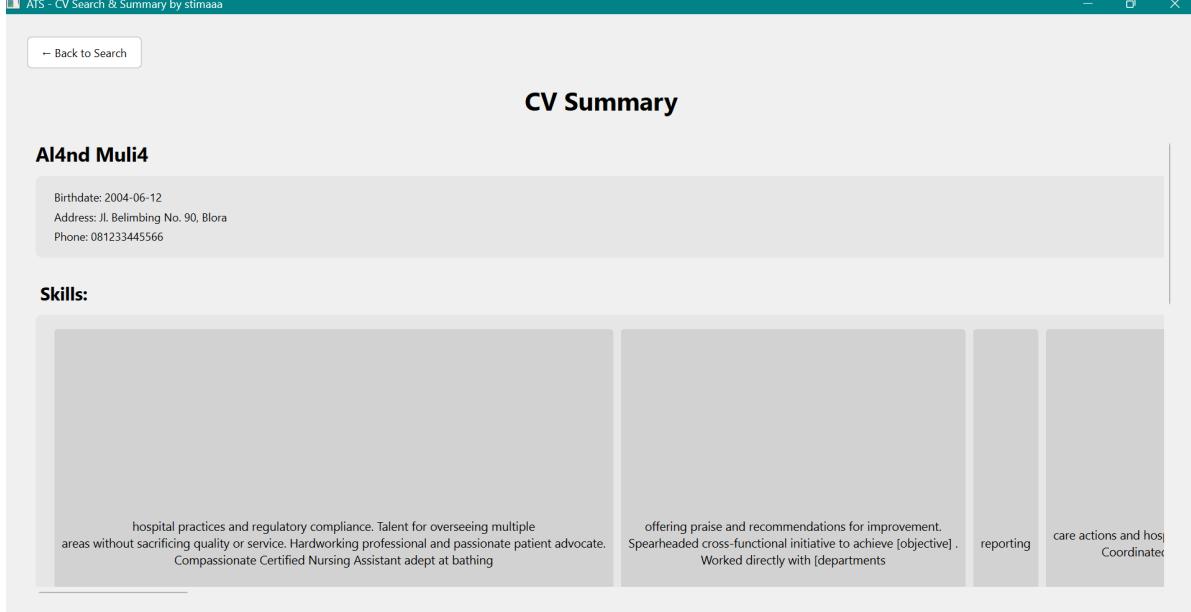
Input
name
Output
<p>The screenshot shows the 'Applicant Tracking System - CV Search' interface. The search bar contains the keyword 'name'. The 'Select Algorithm' dropdown is set to 'KMP'. The search results are displayed in a grid format:</p> <ul style="list-style-type: none"> <li><b>Ikhw4n ALH4KIM</b>: 22 matches. Matched keywords: 1. name: 22 occurrences. Buttons: Summary, View CV.</li> <li><b>Ari3L HerfR150n</b>: 20 matches. Matched keywords: 1. name: 20 occurrences. Buttons: Summary, View CV.</li> <li><b>ar13l h3rfr150n</b>: 14 matches. Matched keywords: 1. name: 14 occurrences. Buttons: Summary, View CV.</li> <li><b>ar13l h3rfr150n</b>: 14 matches. Matched keywords: 1. name: 14 occurrences. Buttons: Summary, View CV.</li> <li><b>H41k4l 455y4uq1</b>: 14 matches. Matched keywords: 1. name: 14 occurrences. Buttons: Summary, View CV.</li> <li><b>Ari3L HerfR150n</b>: 13 matches. Matched keywords: 1. name: 13 occurrences. Buttons: Summary, View CV.</li> <li><b>ALaNd MuLiA</b>: 12 matches. Matched keywords: 1. name: 12 occurrences. Buttons: Summary, View CV.</li> <li><b>4l4nd MuL14</b>: 11 matches. Matched keywords: 1. name: 11 occurrences. Buttons: Summary, View CV.</li> </ul> <p>Exact Search Time (KMP): 404.59 ms   Fuzzy Match: 0.00 ms</p> <p>Total CVs Processed: 600</p>

The screenshot shows a CV summary page for a candidate named 'ALaNd MuLiA'. At the top, there is a back-to-search button and a title 'CV Summary'. Below the title, the candidate's name is displayed in bold. A small box contains personal information: Birthdate: 2003-02-26, Address: Jl. Tomat No. 3, Ambon, and Phone: 081245677899. The section 'Skills:' lists various competencies in a grid. The 'Job History:' section indicates 'No job history extracted.' The 'Education:' section shows a single entry: B (S.E) from Civil Engineering in 1991 at Colorado University, with a graduation year of 2013.

#### 4.3.3. Test Case 3

Input	
name	
Output	
<p>The screenshot shows the 'Applicant Tracking System - CV Search' interface. A search bar contains the keyword 'name'. Below the search bar, it says 'Select Algorithm: Aho-Corasick' (which is highlighted in green). The search results are displayed in a grid of cards:</p> <ul style="list-style-type: none"> <li><b>Ikhw4n ALH4KIM</b>: 22 matches. Matched keywords: 1. name: 22 occurrences. Buttons: Summary, View CV.</li> <li><b>Ari3L HerfR150n</b>: 20 matches. Matched keywords: 1. name: 20 occurrences. Buttons: Summary, View CV.</li> <li><b>ar13l h3rfr150n</b>: 14 matches. Matched keywords: 1. name: 14 occurrences. Buttons: Summary, View CV.</li> <li><b>ar13l h3rfr150n</b>: 14 matches. Matched keywords: 1. name: 14 occurrences. Buttons: Summary, View CV.</li> <li><b>H41k4I 455y4uq1</b>: 14 matches. Matched keywords: 1. name: 14 occurrences. Buttons: Summary, View CV.</li> <li><b>Ari3L HerfR150n</b>: 13 matches. Matched keywords: 1. name: 13 occurrences. Buttons: Summary, View CV.</li> <li><b>ALaNd MuLiA</b>: 12 matches. Matched keywords: 1. name: 12 occurrences. Buttons: Summary, View CV.</li> <li><b>4l4nd MuL14</b>: 11 matches. Matched keywords: 1. name: 11 occurrences. Buttons: Summary, View CV.</li> </ul> <p>At the bottom, there are buttons for 'Previous', 'Page 1 of 1', and 'Next'.</p>	

#### 4.3.4. Test Case 4

Input
Security, info
Output
 <p>The screenshot shows the ATS - CV Search &amp; Summary interface. The search term 'Security, info' is entered in the search bar. The 'Boyer-Moore' algorithm is selected. The results page displays eight search results in cards:</p> <ul style="list-style-type: none"> <li><b>AI4nd Muli4</b>: 39 matches. Matched keywords: 1. Security: 1 occurrence, 2. info: 38 occurrences. Buttons: Summary, View CV.</li> <li><b>AI4nd Muli4</b>: 39 matches. Matched keywords: 1. Security: 1 occurrence, 2. info: 38 occurrences. Buttons: Summary, View CV.</li> <li><b>Ariel Herfrison</b>: 32 matches. Matched keywords: 1. Security: 17 occurrences, 2. info: 15 occurrences. Buttons: Summary, View CV.</li> <li><b>f4rhan naf15</b>: 26 matches. Matched keywords: 1. Security: 7 occurrences, 2. info: 19 occurrences. Buttons: Summary, View CV.</li> <li><b>H41K4L 455YAUQ1</b>: 20 matches. Matched keywords: 1. Security: 7 occurrences. Buttons: Summary, View CV.</li> <li><b>H41K4L 455YAUQ1</b>: 20 matches. Matched keywords: 1. Security: 7 occurrences. Buttons: Summary, View CV.</li> <li><b>4HMAD r4f1</b>: 19 matches. Matched keywords: 1. Security: 2 occurrences. Buttons: Summary, View CV.</li> <li><b>RaDen FrAnC1SCo</b>: 17 matches. Matched keywords: 1. Security: 1 occurrence. Buttons: Summary, View CV.</li> </ul> <p>Exact Search Time (Boyer-Moore): 737.21 ms   Fuzzy Match: 0.00 ms</p> <p>Total CVs Processed: 600</p>
 <p>The screenshot shows the CV Summary interface for the candidate 'AI4nd Muli4'. The summary includes:</p> <ul style="list-style-type: none"> <li><b>Personal Information:</b> Birthdate: 2004-06-12, Address: Jl. Belimbing No. 90, Blora, Phone: 081233445566</li> <li><b>Skills:</b> A section showing four skill categories with descriptions.</li> </ul> <p>Skills listed:</p> <ul style="list-style-type: none"> <li>Offering praise and recommendations for improvement. Spearheaded cross-functional initiative to achieve [objective]. Worked directly with [departments]</li> <li>Reporting</li> <li>Care actions and hospital Coordinated</li> <li>Hospital practices and regulatory compliance. Talent for overseeing multiple areas without sacrificing quality or service. Hardworking professional and passionate patient advocate. Compassionate Certified Nursing Assistant adept at bathing</li> </ul>

#### 4.3.5. Test Case 5

Input
Security, info

**Output**

The screenshot shows the 'ATS - CV Search & Summary by stimaaa' application window. At the top, there's a search bar with placeholder text 'Separate keywords with commas, e.g.: Python, React, SQL'. Below it, a 'Select Algorithm:' dropdown is set to 'KMP', with other options like 'Boyer-Moore' and 'Aho-Corasick' available. To the right, a 'Top matches:' dropdown is set to '8'. The main area displays search results for several names:

- AI4nd Muli4**: 39 matches. Matched keywords: 1. Security: 1 occurrence, 2. info: 38 occurrences. Buttons: Summary, View CV.
- AI4nd Muli4**: 39 matches. Matched keywords: 1. Security: 1 occurrence, 2. info: 38 occurrences. Buttons: Summary, View CV.
- Ariel Herfrison**: 32 matches. Matched keywords: 1. Security: 17 occurrences, 2. info: 15 occurrences. Buttons: Summary, View CV.
- f4rhan naf15**: 26 matches. Matched keywords: 1. Security: 7 occurrences, 2. info: 19 occurrences. Buttons: Summary, View CV.
- H41K4L 455YAUQ1**: 20 matches. Matched keywords: 1. Security: 7 occurrences. Buttons: Summary, View CV.
- H41K4L 455YAUQ1**: 20 matches. Matched keywords: 1. Security: 7 occurrences. Buttons: Summary, View CV.
- 4HMAD r4f1**: 19 matches. Matched keywords: 1. Security: 2 occurrences. Buttons: Summary, View CV.
- RaDen FrAnC1SCo**: 17 matches. Matched keywords: 1. Security: 1 occurrence. Buttons: Summary, View CV.

At the bottom, there are navigation buttons: Previous, Page 1 of 1, and Next.

#### 4.3.6. Test Case 6

**Input**

Security, info

**Output**

The screenshot shows the same application window, but the 'Select Algorithm:' dropdown is now set to 'Aho-Corasick'. The search results are identical to the previous test case, displaying the same names and their respective match counts and details.

#### 4.3.7. Test Case 7

Input
Securiti, infos
Output
<p>The screenshot shows the 'ATS - CV Search &amp; Summary by stimaaa' application. The search bar contains 'Securiti, infos'. The search results are displayed in a grid format:</p> <ul style="list-style-type: none"> <li><b>Ariel Herfrison</b>: 17 matches. Matched keywords: 1. Securiti (typo): 17 occurrences. Buttons: Summary, View CV.</li> <li><b>Mohammad NUGR4H4</b>: 14 matches. Matched keywords: 1. Securiti (typo): 14 occurrences. Buttons: Summary, View CV.</li> <li><b>f4rhan naf15</b>: 12 matches. Matched keywords: 1. Securiti (typo): 7 occurrences, 2. infos: 5 occurrences. Buttons: Summary, View CV.</li> <li><b>F4RH4n Nafis</b>: 8 matches. Matched keywords: 1. Securiti (typo): 8 occurrences. Buttons: Summary, View CV.</li> <li><b>1KHWAN 4IH4KIM</b>: 8 matches. Matched keywords: 1. Securiti (typo): 8 occurrences. Buttons: Summary, View CV.</li> <li><b>1KHWAN 4IH4KIM</b>: 8 matches. Matched keywords: 1. Securiti (typo): 8 occurrences. Buttons: Summary, View CV.</li> <li><b>1KHWAN 4IH4KIM</b>: 8 matches. Matched keywords: 1. Securiti (typo): 8 occurrences. Buttons: Summary, View CV.</li> <li><b>1KHWAN 4IH4KIM</b>: 8 matches. Matched keywords: 1. Securiti (typo): 8 occurrences. Buttons: Summary, View CV.</li> </ul> <p>Search time: 726.97 ms   Fuzzy Match: 5631.90 ms   Total CVs Processed: 600</p>
<p>The screenshot shows the 'CV Summary' page for 'Ariel Herfrison'.</p> <p><b>Personal Information:</b></p> <ul style="list-style-type: none"> <li>Birthdate: 2003-12-10</li> <li>Address: Jl. Durian No. 6, Cirebon</li> <li>Phone: 081277789912</li> </ul> <p><b>Skills:</b></p> <ul style="list-style-type: none"> <li>migration, operating system, Utilities, DHCP, fixed assets, IBM, Internet Explorer, KIX, Adobe Acrobat, MCP, policies, UNIX system, Information Security, Office 97, C</li> </ul> <p><b>Job History:</b></p> <p>No job history extracted.</p> <p><b>Education:</b></p> <ul style="list-style-type: none"> <li>a (nd)</li> <li>- (MSCE)</li> </ul>

#### 4.3.8. Test Case 8

Input
Securiti, infos

## Output

The screenshot shows the 'ATS - CV Search & Summary by stimaaa' application window. The title bar says 'Applicant Tracking System - CV Search'. In the top left, there's a text input field labeled 'Securiti, infos' with the placeholder 'Separate keywords with commas, e.g.: Python, React, SQL'. Below it, a 'Select Algorithm:' dropdown is set to 'KMP', with other options 'Boyer-Moore' and 'Aho-Corasick'. To the right, a 'Top matches:' dropdown is set to '8'. A search bar contains the text 'Search'. Below the search bar, the message 'Exact Search Time (KMP): 2283.16 ms | Fuzzy Match: 5752.93 ms' is displayed, along with 'Total CVs Processed: 600'. The main area displays eight search results in four rows of two columns each. Each result card shows a name, the number of matches, and a list of matched keywords. For example, the first result is 'Ariel Herfrison' with 17 matches, where 'Securiti (typo)' appears 17 times. Each result card has 'Summary' and 'View CV' buttons.

Name	Matches	Matched keywords
Ariel Herfrison	17 matches	1. Securiti (typo): 17 occurrences
Mohammad NUGR4H4	14 matches	1. Securiti (typo): 14 occurrences
f4rhan naf15	12 matches	1. Securiti (typo): 7 occurrences 2. infos: 5 occurrences
F4rH4n Nafis	8 matches	1. Securiti (typo): 8 occurrences
1KHwan 4IH4KIM	8 matches	1. Securiti (typo): 8 occurrences
1KHwan 4IH4KIM	8 matches	1. Securiti (typo): 8 occurrences
1KHwan 4IH4KIM	8 matches	1. Securiti (typo): 8 occurrences
1KHwan 4IH4KIM	8 matches	1. Securiti (typo): 8 occurrences

### 4.3.9. Test Case 9

The screenshot shows the 'ATS - CV Search & Summary by stimaaa' application window. The title bar says 'Applicant Tracking System - CV Search'. In the top left, there's a text input field labeled 'Security, infos. Data, bachelor' with the placeholder 'Separate keywords with commas, e.g.: Python, React, SQL'. Below it, a 'Select Algorithm:' dropdown is set to 'KMP', with other options 'Boyer-Moore' and 'Aho-Corasick'. To the right, a 'Top matches:' dropdown is set to '8'. A search bar contains the text 'Search'. Below the search bar, the message 'Exact Search Time (KMP): 1592.00 ms | Fuzzy Match: 69.91 ms' is displayed, along with 'Total CVs Processed: 600'. The main area displays eight search results in four rows of two columns each. Each result card shows a name, the number of matches, and a list of matched keywords. For example, the first result is 'RAD3N FRANC15CO' with 38 matches, where 'data' appears 38 times. Each result card has 'Summary' and 'View CV' buttons.

Name	Matches	Matched keywords
RAD3N FRANC15CO	38 matches	1. data: 38 occurrences
RAD3N FRANC15CO	38 matches	1. data: 38 occurrences
RAD3N FRANC15CO	38 matches	1. data: 38 occurrences
R4d3N Fr4nc15co	37 matches	1. Security: 3 occurrences 2. data: 34 occurrences
4hmad Rafi	35 matches	1. Security: 6 occurrences 2. bachelor: 2 occurrences 3. data: 27 occurrences
4hmad Rafi	35 matches	1. Security: 6 occurrences 2. bachelor: 2 occurrences 3. data: 27 occurrences
4hmad Rafi	35 matches	1. Security: 6 occurrences 2. bachelor: 2 occurrences 3. data: 27 occurrences

### 4.3.10. Test Case 10

Input
Security, infos, data, bachelor
Output
<p>The screenshot shows the ATS - CV Search &amp; Summary by stimaaa application. The search bar contains the query "Security, infos, data, bachelor". The search algorithm selected is Boyer-Moore. The results section displays four search results in cards:</p> <ul style="list-style-type: none"> <li><b>RAD3N FRANC15CO</b> 38 matches Matched keywords: 1. data: 38 occurrences <a href="#">Summary</a> <a href="#">View CV</a></li> <li><b>RAD3N FRANC15CO</b> 38 matches Matched keywords: 1. data: 38 occurrences <a href="#">Summary</a> <a href="#">View CV</a></li> <li><b>RAD3N FRANC15CO</b> 38 matches Matched keywords: 1. data: 38 occurrences <a href="#">Summary</a> <a href="#">View CV</a></li> <li><b>R4d3N Fr4nc15co</b> 37 matches Matched keywords: 1. Security: 3 occurrences <a href="#">Summary</a> <a href="#">View CV</a></li> </ul> <p>Exact Search Time (Boyer-Moore): 2111.32 ms   Fuzzy Match: 222.42 ms</p> <p>Total CVs Processed: 600</p>

#### 4.3.11. Test Case 11

Input
Security, infos, data, bachelor
Output

The screenshot shows the 'ATS - CV Search & Summary by stimaaa' application window. The title bar reads 'Applicant Tracking System - CV Search'. The search bar contains the keywords 'Security, infos, data, bachelor'. Below the search bar, there are three algorithm selection buttons: 'Select Algorithm:' (highlighted), 'KMP', 'Boyer-Moore', and 'Aho-Corasick' (highlighted). A dropdown menu for 'Top matches' is set to 8. The search results are displayed in a grid of four columns. Each result card shows a name, the number of matches, and a summary of matched keywords. For example, the first row shows 'RAD3N FRANC15CO' with 38 matches, 'Matched keywords: 1. data: 38 occurrences'. The second row shows 'R4d3N Fr4nc15co' with 37 matches, 'Matched keywords: 1. data: 34 occurrences'. The third row shows '4hmad Rafi' with 35 matches, 'Matched keywords: 1. bachelor: 2 occurrences'. The fourth row shows another '4hmad Rafi' entry with 35 matches, 'Matched keywords: 1. bachelor: 2 occurrences'. At the bottom, a message says 'Exact Search Time (Aho-Corasick): 1346.65 ms | Fuzzy Match: 0.00 ms' and 'Total CVs Processed: 600'. Navigation buttons for 'Previous', 'Page 1 of 1', and 'Next' are at the bottom.

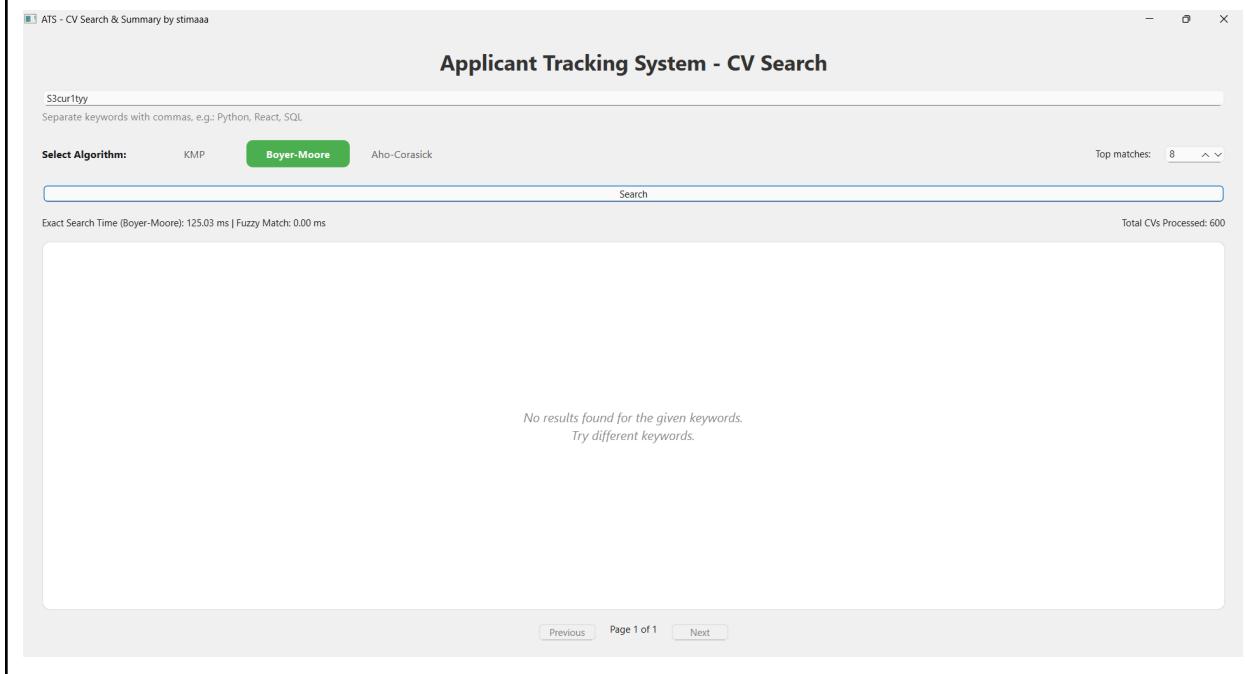
#### 4.3.12. Test Case 12

Input
S3cur1tyy
Output
<p>The screenshot shows the same 'ATS - CV Search &amp; Summary by stimaaa' application window. The search bar now contains 'S3cur1tyy'. The search results grid is empty, and a message in the center of the results area says 'No results found for the given keywords. Try different keywords.' Navigation buttons for 'Previous', 'Page 1 of 1', and 'Next' are at the bottom.</p>

#### 4.3.13. Test Case 13

S3cur1tyy

## Output



### 4.4. Analisis Hasil Pengujian

Algoritma KMP berfokus pada pencarian pola yang cepat dalam teks dengan menghindari pencocokan ulang melalui penggunaan struktur LPS (Longest Prefix Suffix). Fungsi utama dari KMP adalah untuk mempercepat proses pencocokan dengan memanfaatkan pola itu sendiri untuk "melompat" ke posisi yang relevan dalam teks ketika terjadi ketidakcocokan. Algoritma ini efektif saat kita mencari pola yang mungkin muncul beberapa kali dalam teks karena ia memanfaatkan LPS array untuk menghindari pencocokan karakter yang sudah diperiksa sebelumnya. Kelemahan utama dari KMP adalah jika pola yang dicari tidak ada dalam teks, maka algoritma ini tetap melakukan pemrosesan yang cukup intensif, meskipun tidak menghasilkan hasil. Dari test case yang diberikan, algoritma KMP ini memiliki performa yang paling buruk karena pattern dan teks yang digunakan alfabetis sehingga kelebihan dari algoritma ini tidak dimanfaatkan dengan sempurna – KMP lebih cocok untuk pencocokan string biner.

Boyer-Moore adalah algoritma pencarian yang efisien, terutama untuk teks yang sangat panjang, dengan fokus pada dua heuristik utama, yaitu bad character rule dan good suffix rule. Algoritma ini mencoba untuk melompat lebih jauh di dalam teks dengan membandingkan pola dari belakang ke depan. Jika terjadi ketidakcocokan, algoritma ini bisa langsung melompati bagian teks yang sudah diperiksa, yang meningkatkan efisiensinya dibandingkan algoritma lain. Kelebihan dari Boyer-Moore adalah kemampuannya untuk melewati sebagian besar teks ketika terjadi ketidakcocokan, tetapi kelemahannya adalah jika pola terlalu kecil atau jika pencocokan

pertama sulit, ia tidak dapat memberikan banyak keuntungan. Selain itu, jika ada banyak pencocokan di seluruh teks, Boyer-Moore bisa kurang efisien. Sesuai dengan test case yang ada, Boyer-Moore ini merupakan algoritma yang paling efisien untuk segala kondisi – tidak secepat Aho Corasick jika memasukkan banyak info tetapi lebih cepat dari KMP.

Aho-Corasick adalah algoritma pencarian yang sangat efisien untuk mencocokkan banyak pola dalam teks sekaligus. Algoritma ini menggunakan Trie dan fail function untuk melakukan pencocokan. Trie menyimpan setiap pola sebagai jalur dalam pohon, dan fail links memungkinkan pencarian untuk dilanjutkan ke pohon lainnya ketika terjadi ketidakcocokan. Aho-Corasick sangat cocok untuk pencarian banyak pola dalam satu teks, dan sangat efisien dalam menghindari pencocokan ulang. Namun, kelebihannya terletak pada kemampuannya untuk menangani pola tumpang tindih dan melakukan pencarian secara paralel. Sebagai kelemahan, Aho-Corasick bisa menjadi lebih kompleks dalam implementasinya dan membutuhkan lebih banyak memori, terutama jika jumlah pola yang dicocokkan sangat besar.

KMP sangat efisien dalam pencarian pola tunggal dan menghindari pencocokan yang berlebihan melalui LPS. Boyer-Moore sangat efisien untuk pencarian pola yang jarang muncul dalam teks, berkat kemampuannya untuk melompat lebih jauh. Aho-Corasick sangat efisien dalam mencari banyak pola secara bersamaan, meskipun dapat lebih kompleks dan membutuhkan lebih banyak memori.

## **BAB V**

### **KESIMPULAN, SARAN, DAN REFLEKSI**

#### **5.1. Kesimpulan**

Boyer-Moore, Knuth-Morris-Pratt (KMP), dan Aho-Corasick adalah algoritma pencarian string fundamental yang masing-masing menawarkan pendekatan unik untuk menemukan pola dalam teks. Boyer-Moore efektif untuk pencarian pola tunggal dalam teks besar, seringkali menjadi yang tercepat dalam praktiknya karena heuristik bad character dan good suffix yang memungkinkannya "melompati" bagian teks. KMP menjamin kinerja linear  $O(n+m)$  yang stabil untuk pencarian pola tunggal dengan menghindari backtracking melalui penggunaan tabel prefiks yang telah dihitung. Sementara itu, Aho-Corasick unggul dalam skenario pencarian multi-pola, secara efisien menemukan semua kemunculan banyak pola dalam satu pass melalui teks dengan memanfaatkan struktur data automaton, menjadikannya ideal untuk tugas seperti pencarian keyword berganda dalam sistem pelacakan pelamar (ATS) Anda.

#### **5.2. Saran**

Sistem ATS ini merupakan langkah maju dalam otomatisasi rekrutmen. Untuk pengembangannya di masa depan, fokus pada validasi dan keragaman dataset CV agar representasi dunia nyata lebih akurat, serta optimasi performa algoritma lebih lanjut di bawah skenario beban tinggi. Pertimbangkan juga untuk memperluas fungsionalitas Regex agar lebih tangguh terhadap berbagai format CV yang tidak standar.

#### **5.3. Refleksi**

Tugas besar ini memberikan pengalaman berharga dalam mengintegrasikan berbagai konsep seperti pattern matching, basis data, dan GUI. Tantangan utama adalah ekstraksi data yang konsisten dari berbagai format PDF CV serta penyesuaian algoritma Levenshtein Distance untuk akurasi optimal. Meskipun demikian, proyek ini berhasil menunjukkan potensi otomatisasi penyaringan CV dan mendorong kami untuk terus belajar dalam pengembangan sistem yang kompleks.

## DAFTAR PUSTAKA

- [1] <https://docs.google.com/document/d/1aVJtQ9oj-WE7GrQwaHv3xzYcAMIsPBLAX4SLBcPZQA/edit?tab=t.0>
- [2] [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)
- [3] <https://www.rippling.com/glossary/ats>
- [4] <https://www.thebalancemoney.com/cv-samples-and-writing-tips-2060349>

## LAMPIRAN

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 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 applicant.	✓	
10.	Membuat bonus algoritma Aho-Corasick.	✓	
11.	Membuat bonus video dan diunggah pada Youtube.	✓	

Berikut lampiran tautan.

Repository GitHub	<a href="https://github.com/shanlie20/Tubes3_stimaaa">https://github.com/shanlie20/Tubes3_stimaaa</a>
Link YouTube	<a href="https://www.youtube.com/watch?v=NvXLgeCFD2I">https://www.youtube.com/watch?v=NvXLgeCFD2I</a>