

# **LAPORAN TUGAS BESAR IF2211**

## **STRATEGI ALGORITMA**

Tugas Besar 3 Strategi Algoritma



Kelompok CVRobin

Abrar Abhirama W. 13523038

Brian Albar Hadian 13523048

Faqih M. Syuhada 13523057

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**JL. GANESA 10, BANDUNG 40132**

**2025**

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>BAB I.....</b>	<b>3</b>
<b>BAB II.....</b>	<b>7</b>
2.1. Pattern Matching (Exact Matching).....	7
2.1.1. Regular Expression (Regex).....	7
2.1.2. Knuth-Morris-Pratt (KMP).....	7
2.1.3 Boyer Moore (BM).....	8
2.2. Dasar Teori Fuzzy Matching.....	9
<b>BAB III.....</b>	<b>10</b>
3.1. Langkah Penyelesaian Masalah.....	10
3.2. Mapping Persoalan CV ATS.....	13
3.3. Fungsionalitas dan Arsitektur Aplikasi.....	15
<b>BAB IV.....</b>	<b>17</b>
4.1. Spesifikasi Teknis Program.....	17
4.1.1. Struktur Data.....	17
4.1.2. Fungsi dan Prosedur.....	17
4.2. Penjelasan Tata Cara Penggunaan Program.....	19
4.2.1. Mencari pelamar kerja sesuai dengan masukan kata kunci.....	19
4.2.2. Melihat semua pelamar kerja.....	19
4.2.3. Melihat semua pekerjaan yang mungkin.....	19
4.3. Pengujian.....	20
4.4. Analisis dan Pembahasan.....	21
<b>BAB V.....</b>	<b>23</b>
5.1. Kesimpulan.....	23
5.2. Saran.....	23
5.3. Komentar.....	23
5.4. Refleksi.....	24
<b>LAMPIRAN.....</b>	<b>25</b>
<b>DAFTAR PUSTAKA.....</b>	<b>26</b>

## BAB I

### DESKRIPSI TUGAS

CV Applicant Tracking System (CV ATS) merupakan aplikasi yang berperan untuk meringankan beban departemen talenta ketika melakukan rekrutmen baru. CV ATS dirancang untuk mempermudah perusahaan dalam menyaring dan mencocokkan informasi kandidat dari berkas lamaran, khususnya Curriculum Vitae (CV). Selain itu, ATS memungkinkan perusahaan

untuk mengelola ribuan dokumen lamaran secara otomatis dan memastikan kandidat yang relevan dapat ditemukan dengan cepat.

Untuk suatu CV, CV ATS akan melakukan analisis terhadap data yang ada pada CV tersebut dan mengembalikan hasil dalam format yang dapat dibaca dengan cepat dan dapat diproses lebih lanjut untuk memperoleh informasi lainnya, seperti kategori pekerjaan sebelumnya, tingkat pendidikan terakhir, dsb. Dengan adanya CV ATS, diharapkan bahwa proses rekrutmen dapat dilakukan dengan lebih cepat dan memberikan ruang untuk pengembangan di bagian lain pada departemen tersebut.

Pada Tugas Besar ketiga Strategi Algoritma ini, mahasiswa diminta untuk mengimplementasikan CV ATS yang menggunakan pattern matching (exact matching) Boyer-Moore, Knuth-Morris-Pratt, dan Aho-Corasick untuk melakukan exact matching. Selain itu, mahasiswa juga diminta untuk menerapkan fuzzy matching dengan menggunakan levenshtein distance.

Komponen-komponen utama dari aplikasi ini antara lain:

1. Penerapan exact matching menggunakan pattern matching (BM, KMP, dan AH)
2. Penerapan fuzzy matching menggunakan levenshtein distance
3. Penerapan regex untuk memperoleh bagian-bagian dari CV yang diberikan
4. Penerapan aplikasi dalam bentuk GUI
5. Penerapan database untuk menyimpan metadata dari pelamar kerja dan CV yang terkait dengan pelamar tersebut

## Spesifikasi Wajib

- Sistem yang dibangun pada tugas besar ini bertujuan untuk melakukan pencocokan dan pencarian data pelamar kerja berbasis CV yang diunggah oleh pengguna. Sistem dikembangkan menggunakan bahasa pemrograman Python dengan antarmuka desktop berbasis pustaka seperti Tkinter, PyQt, atau framework lain yang relevan. Dalam proses pencocokan kata kunci, sistem wajib mengimplementasikan algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Untuk mengukur kemiripan saat terjadi kesalahan input atau perbedaan penulisan, sistem juga menerapkan algoritma Levenshtein Distance. Selain itu, Regular Expression (Regex) digunakan untuk mengekstrak informasi penting dari teks CV secara otomatis. Oleh karena itu, penguasaan pengembangan GUI dan pemrosesan string di Python sangat penting untuk menyelesaikan tugas ini secara optimal.
- Program yang dikembangkan harus menggunakan basis data berbasis MySQL untuk menyimpan informasi hasil ekstraksi dari CV yang telah diunggah. Basis data akan menyimpan informasi berupa profil pelamar beserta lokasi penyimpanan file CV di dalam sistem.

- Fitur utama dari sistem ini adalah kemampuannya untuk ekstraksi teks dari CV dalam format PDF. Setelah dokumen CV diunggah, program harus mampu melakukan ekstraksi teks secara otomatis dan mengubahnya menjadi profil pelamar kerja. Profil tersebut akan ditampilkan kepada pengguna tanpa perlu intervensi manual tambahan. Proses ini akan membantu mempercepat identifikasi dan penilaian awal terhadap pelamar.
- Sistem wajib menyediakan fitur pencarian terhadap data pelamar menggunakan kata kunci atau kriteria tertentu yang ditentukan oleh pengguna (misalnya nama, skill tertentu, atau pengalaman kerja). Pencarian ini akan dilakukan terhadap semua data dalam database dan bertujuan untuk menemukan pelamar yang paling relevan dengan kriteria pencarian tersebut. Proses pencarian dilakukan sepenuhnya secara in-memory agar hasilnya cepat dan responsif. Proses pencarian utamanya dilakukan secara exact matching.
- Setelah exact matching, apabila tidak ditemukan kecocokan secara persis, sistem harus melakukan fuzzy matching. Untuk setiap kata kunci yang tidak ditemukan satupun kemunculan saat exact matching, lakukan pencarian kembali dengan perhitungan tingkat kemiripan menggunakan algoritma Levenshtein Distance. Algoritma ini memungkinkan sistem untuk tetap menampilkan hasil pencarian yang relevan, meskipun terdapat perbedaan minor atau kesalahan ketik pada input pengguna. Hal ini sangat membantu pengguna untuk tetap mendapatkan hasil terbaik tanpa harus memasukkan kata kunci secara sempurna.
- Apabila salah satu hasil pencarian di-klik, sistem harus dapat menampilkan ringkasan/summary dari lamaran tersebut. Pada halaman ringkasan, harus terdapat opsi (e.g. tombol) untuk melihat CV secara keseluruhan.
- Informasi yang ditampilkan dalam ringkasan/summary CV dari hasil pencarian harus mencakup data penting dari pelamar, yaitu identitas (nama, kontak, dan informasi pribadi lainnya) yang diperoleh dari basis data. Kemudian terdapat beberapa data yang diperoleh dengan cara ekstraksi melalui regular expression, meliputi:
  - Ringkasan pelamar (summary/overview)
  - Keahlian pelamar (skill)
  - Pengalaman kerja (e.g. tanggal dan jabatan)
  - Riwayat pendidikan (e.g. tanggal kelulusan, universitas, dan gelar)
- Dengan menampilkan informasi-informasi penting tersebut, sistem dapat memberikan ringkasan profil yang relevan kepada pengguna.

- Pengguna aplikasi dapat memilih algoritma pencocokan yang ingin digunakan untuk exact matching, yaitu antara KMP atau BM, (bisa pula Aho-Corasick apabila mengerjakan bonus), sebelum memulai proses pencarian. Pilihan algoritma ini akan mempengaruhi cara sistem memindai dan mencocokkan kata kunci dengan isi CV. Hal ini memberikan fleksibilitas dan pemahaman algoritmik yang lebih luas bagi pengguna atau pengembang.
- Pengguna aplikasi dapat menentukan jumlah CV yang ditampilkan. CV yang ditampilkan diurutkan mulai dari CV dengan jumlah kecocokan kata kunci terbanyak.
- Setelah pencarian CV, sistem akan menampilkan waktu pencarian. Terdapat 2 waktu berbeda yang perlu ditampilkan. Pertama adalah waktu pencarian exact match menggunakan algoritma KMP atau BM. Kemudian, tampilkan juga waktu pencarian fuzzy match menggunakan Levenshtein Distance apabila terdapat kata kunci yang belum ditemukan.
- Aplikasi yang dibuat harus memiliki antarmuka pengguna (user interface) yang intuitif dan menarik, sehingga mudah digunakan bahkan oleh pengguna awam. Komponen-komponen penting seperti, input keyword, pemilihan algoritma, serta hasil pencarian harus disusun dengan jelas dan rapi. Pengembang juga diperkenankan menambahkan fitur tambahan yang dapat memperkaya fungsi dan pengalaman pengguna, sebagai bentuk kreativitas dan inisiatif dalam mengembangkan sistem yang lebih bermanfaat dan inovatif.

### Spesifikasi Bonus

- Enkripsi Data Profil Applicant (maksimal 5 poin)
  - Lakukan proses enkripsi terhadap data profil applicant yang disimpan di dalam basis data untuk menjaga kerahasiaan informasi pribadi pelamar kerja. Enkripsi ini perlu diterapkan agar data tetap aman meskipun terjadi akses langsung ke basis data. Implementasi tidak diperkenankan menggunakan pustaka enkripsi bawaan Python (cryptography atau hashlib), semakin kompleks dan aman skema enkripsi yang dibuat maka potensi nilai bonus pun akan semakin tinggi.
- Implementasi Algoritma Aho-Corasick (maksimal 10 poin)
  - Tambahkan dukungan algoritma Aho-Corasick sebagai alternatif metode pencarian kata kunci yang efisien untuk multi-pattern matching. Dengan algoritma ini, sistem dapat mencocokkan seluruh daftar keyword sekaligus dalam satu proses traversal teks, sehingga lebih cepat dibandingkan pendekatan konvensional satu per satu.

Kehadiran opsi algoritma ini menunjukkan pemahaman lanjutan terhadap strategi optimasi pencarian string.

- Pembuatan Video Aplikasi (maksimal 5 poin)
  - Buatlah video presentasi mengenai aplikasi yang telah dikembangkan, mencakup penjelasan fitur-fitur utama serta demonstrasi penggunaannya. Video harus menyertakan audio narasi dan menampilkan wajah dari seluruh anggota kelompok. Kualitas penyampaian dan visual akan mempengaruhi penilaian. Upload video ke YouTube dan pastikan video dapat diakses publik. Sebagai referensi, Anda dapat melihat video tugas besar dari tahun-tahun sebelumnya dengan kata kunci seperti “Tubes Stima”, “Tugas Besar Stima”, atau “Strategi Algoritma”.

## BAB II

### LANDASAN TEORI

#### 2.1. Pattern Matching (Exact Matching)

Pattern matching atau pencocokan pola adalah proses untuk menemukan semua kemunculan sebuah pattern (pola) P dengan panjang m di dalam sebuah text (teks) T dengan panjang n. Asumsi dasarnya adalah panjang pola jauh lebih kecil daripada panjang teks ( $m \ll n$ ). Tujuan utamanya adalah untuk menemukan lokasi atau indeks pertama di mana pola P bersesuaian dengan bagian dari teks T.

Pencocokan pola memiliki banyak aplikasi dalam dunia nyata, seperti fitur "find" pada editor teks , mesin pencari web seperti Google , analisis citra untuk pengenalan objek , hingga pencocokan rantai asam amino dalam bioinformatika.

##### 2.1.1. Regular Expression (Regex)

Regular Expression (Regex) adalah sebuah formalisme bahasa atau notasi khusus untuk mendefinisikan sebuah pola pencarian string. Regex memungkinkan pencocokan string yang lebih fleksibel daripada pencocokan harfiah (exact matching) karena dapat mendefinisikan aturan-aturan kompleks, seperti karakter yang boleh muncul, jumlah kemunculan, dan variasinya dalam satu baris ekspresi.

Regex book	Version History	Feedback	Blog
	Options	Quick Reference	
19 t t ntr	<ul style="list-style-type: none"> <li>.</li> <li>\.</li> <li>^</li> <li>\$</li> <li>\d,\w,\s</li> <li>\D,\W,\S</li> <li>[abc]</li> <li>[a-z]</li> <li>[^abc]</li> <li>aa bb</li> <li>?</li> <li>*</li> <li>+</li> <li>{n}</li> <li>{n,}</li> <li>{m,n}</li> <li>??,*?,+?,{n}?, etc.</li> <li>(expr)</li> <li>(?:expr)</li> <li>(?=expr)</li> <li>(?!expr)</li> </ul>	<ul style="list-style-type: none"> <li>Any character except newline.</li> <li>A period (and so on for \*, \(), \\\, etc.)</li> <li>The start of the string.</li> <li>The end of the string.</li> <li>A digit, word character [A-Za-z0-9_], or whitespace.</li> <li>Anything except a digit, word character, or whitespace.</li> <li>Character a, b, or c.</li> <li>a through z.</li> <li>Any character except a, b, or c.</li> <li>Either aa or bb.</li> <li>Zero or one of the preceding element.</li> <li>Zero or more of the preceding element.</li> <li>One or more of the preceding element.</li> <li>Exactly n of the preceding element.</li> <li>n or more of the preceding element.</li> <li>Between m and n of the preceding element.</li> <li>Same as above, but as few as possible.</li> <li>Capture expr for use with \1, etc.</li> <li>Non-capturing group.</li> <li>Followed by expr.</li> <li>Not followed by expr.</li> </ul>	<span style="font-size: small;">⊕</span> <span style="font-size: small;">✖</span>
		<a href="#">Near-complete reference</a>	

Gambar ....

Regex digunakan dengan menyusun notasi-notasi di atas untuk membentuk pola yang diinginkan. Contohnya, untuk menemukan kata yang diawali dengan huruf kapital, kita bisa menggunakan [A-Z][a-z]\*. Pola ini mencari sebuah huruf besar ([A-Z]) yang diikuti oleh nol atau lebih huruf kecil ([a-z]\*).

### 2.1.2. Knuth-Morris-Pratt (KMP)

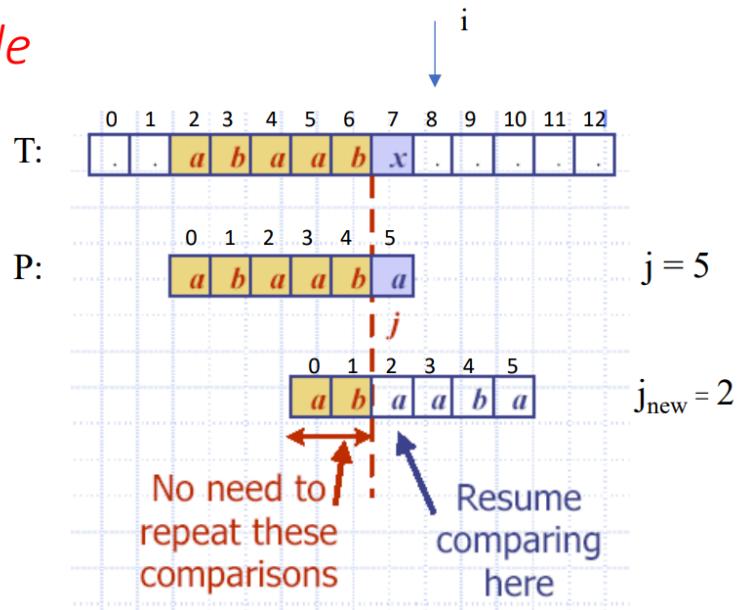
Algoritma Knuth-Morris-Pratt (KMP) adalah salah satu algoritma pencocokan string yang efisien. Berbeda dengan algoritma Brute Force yang menggeser pola satu karakter setiap kali terjadi ketidakcocokan, KMP menggunakan informasi dari pola itu sendiri untuk melakukan pergeseran yang lebih "cerdas" dan menghindari perbandingan yang tidak perlu.

Kunci dari algoritma KMP adalah fungsi pinggiran (border function) atau yang juga dikenal sebagai failure function  $b(k)$ . Fungsi ini melakukan pra-pemrosesan pada pola P untuk menemukan panjang prefiks terpanjang dari  $P[0..k]$  yang juga merupakan sufiks dari  $P[1..k]$ .

Ketika terjadi ketidakcocokan antara teks  $T[i]$  dan pola  $P[j]$ , algoritma tidak langsung menggeser pola sejauh satu karakter. Sebaliknya, pola digeser ke kanan berdasarkan nilai dari fungsi pinggiran. Hal ini memungkinkan algoritma untuk melanjutkan pencocokan tanpa harus membandingkan ulang karakter-karakter pada teks yang sudah pasti cocok.

Misalkan kita memiliki teks T dan pola P. Jika ketidakcocokan terjadi pada  $P[j]$ , kita akan melihat nilai  $b(k)$  di mana  $k = j-1$ . Pergeseran dilakukan sehingga prefiks dari P yang cocok dengan sufiks P akan selaras dengan teks, dan perbandingan dapat dilanjutkan dari titik baru.

### Example



Total kompleksitas waktu algoritma KMP adalah  $O(m+n)$ , yang jauh lebih cepat dibandingkan kasus terburuk *Brute Force* ( $O(mn)$ ).

#### 2.1.3 Boyer Moore (BM)

Algoritma Boyer-Moore (BM) adalah algoritma pencocokan string lain yang sangat efisien dan dalam praktiknya sering kali menjadi yang tercepat. Algoritma ini menggunakan dua teknik utama: looking-glass technique dan character-jump technique.

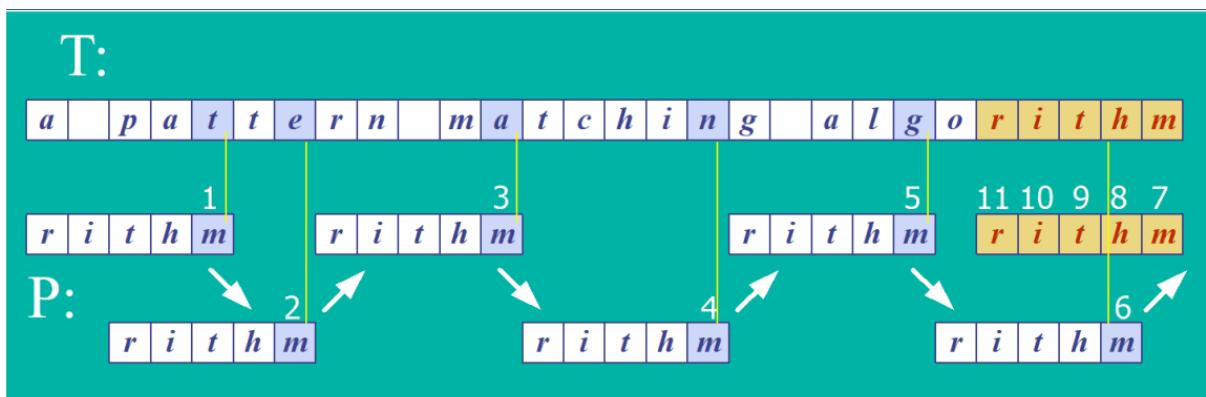
Looking-Glass Technique: Tidak seperti KMP, BM membandingkan pola P dengan teks T dari kanan ke kiri (mundur), dimulai dari karakter terakhir P.

Character-Jump Technique: Ketika terjadi ketidakcocokan antara karakter di teks  $T[i]$  (sebut saja x) dan karakter di pola  $P[j]$ , BM melakukan pergeseran besar ke kanan. Pergeseran ini didasarkan pada dua heuristik:

Bad Character Heuristic: Jika karakter x dari teks yang menyebabkan mismatch ada di dalam pola P, pola akan digeser ke kanan sehingga kemunculan terakhir x di P sejajar dengan x di T.

Good Suffix Heuristic: Jika ada sebagian dari pola yang sudah cocok dengan teks (sufiks yang baik), pola digeser untuk menyalaraskan kemunculan berikutnya dari sufiks tersebut di dalam pola itu sendiri.

Untuk mengimplementasikan Bad Character Heuristic, BM melakukan pra-pemrosesan pada pola untuk membuat last occurrence function  $L(x)$ . Fungsi ini menyimpan indeks kemunculan terakhir dari setiap karakter x yang ada di dalam alfabet pada pola P. Jika karakter x tidak ada di P, pola dapat digeser melewati posisi x sepenuhnya.



Dalam banyak kasus, terutama dengan alfabet yang besar (seperti teks bahasa Inggris), Boyer-Moore secara signifikan lebih cepat daripada *Brute Force* dan KMP.

#### 2.1.4 Aho Corasick (AH)

Algoritma Aho-Corasick adalah ekstensi dari algoritma KMP yang dirancang untuk mencari semua kemunculan dari sekumpulan pola (sebuah kamus kata kunci) di dalam sebuah teks secara bersamaan dalam satu kali proses. Algoritma ini sangat efisien dan

banyak digunakan dalam aplikasi seperti pemfilteran konten, spell-checking, dan deteksi malware.

**Membangun Trie (Finite Automaton):** Langkah pertama adalah membangun sebuah finite automaton keadaan terbatas dari semua pola yang ingin dicari. Struktur data yang umum digunakan untuk ini adalah Trie (juga dikenal sebagai prefix tree). Setiap simpul di Trie merepresentasikan sebuah prefiks, dan setiap kata kunci dalam kamus akan berakhir pada sebuah simpul yang ditandai sebagai output.

**Menambahkan Failure Links:** Mirip dengan border function pada KMP, Aho-Corasick menambahkan "link kegagalan" (failure links) pada setiap simpul di Trie. Jika saat memproses teks, karakter berikutnya tidak memiliki transisi yang valid dari simpul saat ini, algoritma akan mengikuti failure link ke simpul lain yang merepresentasikan prefiks terpanjang yang juga merupakan sufiks dari string yang sudah diproses. Ini memungkinkan pencarian untuk dilanjutkan tanpa harus memulai dari awal.

**Memproses Teks:** Teks di-scan satu kali dari kiri ke kanan. Dimulai dari akar Trie, algoritma mengikuti transisi berdasarkan karakter teks. Setiap kali sebuah simpul tercapai, algoritma memeriksa apakah simpul tersebut (atau simpul yang terhubung melalui failure links) ditandai sebagai output. Jika ya, maka sebuah pola telah ditemukan. Misalkan kita ingin mencari kata kunci {he, she, his, hers} dalam teks ushers.

**Trie Dibangun:** Sebuah Trie dibangun dari kata-kata kunci tersebut.

**Failure Links Dibuat:** Failure links ditambahkan. Misalnya, dari simpul yang merepresentasikan "sh", failure link-nya akan menunjuk ke simpul yang merepresentasikan "h" (karena "h" adalah prefiks terpanjang dari "sh" yang juga ada di kamus).

Pencocokan:

- u: Tidak ada transisi, tetap di akar.
- s: Tidak ada transisi, tetap di akar.
- h: Pindah ke simpul "h".
- e: Pindah ke simpul "he". Simpul ini adalah output, jadi kita menemukan "he".
- r: Tidak ada transisi dari "he". Ikuti failure link ke akar. Dari akar, r tidak ada transisi.
- s: Pindah ke simpul "s". Lalu ikuti failure link "s" ke "h" (dari "sh"), lalu ke "she" (output). Kita menemukan "she" dan "he".

Kompleksitas algoritma Aho-Corasick adalah  $O(n + m + z)$ , di mana  $n$  adalah panjang teks,  $m$  adalah total panjang semua pola, dan  $z$  adalah jumlah total kemunculan pola di dalam teks. Ini membuatnya sangat efisien untuk pencarian multi-pola.

## 2.2. Dasar Teori Fuzzy Matching

Berbeda dengan exact matching yang mencari kesamaan persis, fuzzy matching adalah teknik yang digunakan untuk menemukan string yang "cukup mirip" atau mendekati pola yang dicari. Dalam konteks sistem ATS, ini sangat berguna untuk mengatasi kesalahan pengetikan (typo), variasi ejaan, atau singkatan yang mungkin ada pada CV atau pada input kata kunci dari pengguna. Salah satu algoritma paling populer untuk mengukur kemiripan ini adalah Levenshtein Distance.

### 2.2.1 Levenshtein Distance

Algoritma Levenshtein Distance menghitung "jarak" antara dua string. Jarak ini didefinisikan sebagai jumlah minimum operasi penyuntingan satu karakter yang diperlukan untuk mengubah string pertama menjadi string kedua. Terdapat tiga jenis operasi yang diizinkan:

- Insersi (Insertion): Menambahkan satu karakter ke dalam string.
- Delesi (Deletion): Menghapus satu karakter dari string.
- Substitusi (Substitution): Mengganti satu karakter dengan karakter lain.

Perhitungan jarak ini umumnya dilakukan menggunakan pendekatan Dynamic Programming. Sebuah matriks berukuran  $(m+1) \times (n+1)$  dibangun, di mana  $m$  dan  $n$  adalah panjang dari kedua string. Setiap sel pada matriks,  $D[i][j]$ , menyimpan jarak Levenshtein antara  $i$  karakter pertama dari string pertama dan  $j$  karakter pertama dari string kedua.

Nilai setiap sel dihitung berdasarkan nilai dari sel-sel tetangganya:

$$\text{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \text{lev}_{a,b}(i-1, j) + 1 \\ \text{lev}_{a,b}(i, j-1) + 1 \\ \text{lev}_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Di mana cost bernilai 0 jika karakter ke- $i$  dan ke- $j$  sama, dan 1 jika berbeda. Nilai akhir, yaitu  $D[m][n]$ , adalah jarak Levenshtein total antara kedua string tersebut.

Dalam proyek ini, ketika pencarian exact match tidak menemukan kata kunci, sistem akan melakukan fuzzy matching dengan menghitung jarak Levenshtein antara kata kunci tersebut dengan setiap kata dalam teks CV. Jika jaraknya berada di bawah ambang batas (threshold) yang telah ditentukan, kata tersebut dianggap sebagai kecocokan.

## 2.3. Enkripsi Data

Untuk menjaga kerahasiaan informasi pribadi pelamar kerja, data yang disimpan di dalam basis data MySQL dienkripsi. Spesifikasi tugas mensyaratkan implementasi skema enkripsi tanpa menggunakan pustaka bawaan Python seperti cryptography atau hashlib. Oleh karena itu, dikembangkanlah sebuah skema enkripsi kustom yang terdiri dari dua komponen utama: stream cipher berbasis LFSR untuk kerahasiaan dan Custom HMAC untuk integritas data.

### 2.3.1. Kerahasiaan dengan Stream Cipher dan LFSR

Stream cipher adalah metode enkripsi simetris di mana setiap bit atau byte dari plaintext dienkripsi satu per satu. Ini dilakukan dengan menggabungkan plaintext dengan aliran data pseudorandom yang disebut keystream, umumnya menggunakan operasi XOR.

Linear Feedback Shift Register (LFSR): Dalam implementasi ini, keystream dibangkitkan menggunakan LFSR. LFSR adalah sebuah register geser yang bit inputnya merupakan fungsi linear dari keadaan sebelumnya. Dengan seed (kunci) dan posisi tap yang telah ditentukan, LFSR dapat menghasilkan urutan bit yang panjang dan tampak acak, namun sepenuhnya deterministik. Urutan bit inilah yang menjadi keystream.

Proses Enkripsi/Dekripsi:

- Enkripsi:  $\text{Ciphertext} = \text{Plaintext} \oplus \text{Keystream}$
- Dekripsi:  $\text{Plaintext} = \text{Ciphertext} \oplus \text{Keystream}$

Prosesnya identik karena sifat operasi XOR. Selama kunci (seed LFSR) yang digunakan sama, keystream yang sama dapat dibangkitkan untuk proses dekripsi.

### 2.3.2. Integritas Data dengan Custom HMAC

Enkripsi hanya menjaga kerahasiaan data, namun tidak melindunginya dari modifikasi atau perusakan (tampering). Untuk memastikan bahwa data tidak diubah sejak dienkripsi, digunakan Message Authentication Code (MAC).

- HMAC (Hash-based MAC): HMAC adalah jenis MAC spesifik yang menggunakan fungsi hash kriptografis. Karena pustaka hash standar dilarang, sebuah fungsi hash sederhana namun efektif dibuat secara kustom.
- Proses Otentikasi: Sebuah tag otentikasi dibuat dengan menggabungkan kunci rahasia dengan ciphertext menggunakan fungsi hash kustom. Tag ini kemudian digabungkan dengan ciphertext sebelum disimpan ke database.
- Proses Verifikasi: Saat dekripsi, tag dihitung ulang dari ciphertext yang diterima menggunakan kunci yang sama. Jika tag yang dihitung ulang tidak cocok dengan tag yang diterima, berarti data telah diubah atau kunci yang digunakan salah. Proses dekripsi akan dibatalkan untuk mencegah penggunaan data yang tidak valid.

Dengan menggabungkan LFSR untuk enkripsi dan Custom HMAC untuk otentikasi, skema ini menyediakan baik kerahasiaan maupun integritas untuk data sensitif pelamar.

## BAB III

### ANALISIS PEMECAHAN MASALAH

#### 3.1. Langkah Penyelesaian Masalah

Langkah penyelesaian penerapan CV ATS dilakukan secara bertahap dengan tahapan sebagai berikut.

##### 1. Identifikasi fungsionalitas dasar dan fitur utama

Identifikasi fungsionalitas dasar dilakukan dengan uji literasi terhadap spesifikasi yang telah diberikan dan menghubungkan setiap fitur terhadap data masukan ataupun dengan fitur lainnya. Selain itu, dilakukan juga identifikasi terhadap tanggung jawab untuk setiap fitur yang akan diimplementasikan.

Berdasarkan definisi tersebut, diperoleh fitur utama beserta dengan fungsionalitas dasar sebagai berikut.

###### a. GUI

- i. Menjadi antarmuka untuk pengguna ketika memasukkan kata kunci, algoritma yang ingin digunakan, dan jumlah CV yang ingin ditampilkan
- ii. Menjadi antarmuka untuk sistem ketika menampilkan hasil dari proses *pattern matching*

###### b. Penyimpanan metadata melalui basis data relasional

- i. Menyimpan seluruh metadata pelamar kerja dalam format nama, tanggal lahir, alamat, nomor telepon, alamat CV, dan pekerjaan yang ingin dilamar
- c. Melakukan pencarian data terhadap kata kunci yang diberikan, baik exact matching ataupun fuzzy matching
  - i. Melakukan pencarian data terhadap kata kunci yang diberikan secara exact matching melalui algoritma KMP, BM, dan AH
  - ii. Melakukan pencarian data terhadap kata kunci yang diberikan secara *fuzzy matching* melalui pendekatan *levenshtein distance*
- d. Melakukan pemrosesan data pada CV untuk kata kunci yang diberikan untuk segmentasi CV

- i. Melakukan pemrosesan data pada CV untuk kata kunci yang tertentu untuk memperoleh rangkuman dan bagian-bagian esensial

2. Identifikasi struktur data, prosedur dan fungsi berdasarkan langkah (1)

Selanjutnya, berdasarkan fitur-fitur tersebut, didefinisikan struktur data, fungsi, dan prosedur dengan tanggung jawab sebagai berikut.

- a. Struktur data

- i. CV

1. Menyimpan CV dalam format *string*

- ii. Page

1. Menampilkan GUI

- iii. PageComponent

1. Menampilkan hasil dalam bentuk komponen pada GUI

- b. Fungsi dan Prosedur

- i. start()

1. Menjalankan program utama, menginisialisasi GUI, dan mengaktifkan basis data

- ii. setup()

1. Melakukan pengisian awal terhadap basis data dan menginisialisasi GUI

- iii. getDataFromDatabase() :

1. Melakukan pengambilan data terhadap basis data sesuai dengan konteks program tertentu

- iv. ahoCorasick()

1. Mengimplementasikan algoritma Aho-Corasick untuk *exact matching*

- v. knuthMorrisPratt()

1. Mengimplementasikan algoritma Knuth-Morris-Pratt untuk *exact matching*

- vi. boyerMoore()

1. Mengimplementasikan algoritma Boyer-Moore untuk *exact matching*
  - vii. levenshteinDistance()
    1. Mengimplementasikan pendekatan Levenshtein Distance untuk *fuzzy matching*
  - viii. fuzzyMatching()
    1. Mengimplementasikan levenshteinDistance untuk CV sedemikian sehingga diperoleh hasil untuk kata kunci yang diberikan
  - ix. preprocessCV()
    1. Melakukan pemrosesan CV untuk memperoleh rangkuman dari CV berdasarkan alamat CV
  - x. encrypt()
    1. Mengimplementasikan enkripsi LFSR untuk data pada basis data
  - xi. decrypt()
    1. Mengimplementasikan dekripsi LFSR untuk data pada basis data
3. Identifikasi alur program
- Kemudian, berdasarkan fungsi dan prosedur yang telah diperoleh, didefinisikan setiap fitur program dengan alur program sebagai berikut.
- a. Mencari pelamar kerja sesuai dengan masukan kata kunci
    - i. Pada laman Home, ketik kata kunci pada kolom kata kunci dan tekan 'Enter' setiap selesai memasukkan kata kunci
    - ii. Pilih algoritma yang ingin digunakan untuk mencari kata kunci tersebut
    - iii. Pilih jumlah CV yang ingin ditampilkan
    - iv. Sistem akan membawa Anda menuju laman Hasil yang mengembalikan metadata pelamar kerja serta jumlah ditemukannya kata kunci untuk setiap pelamar kerja
  - b. Melihat semua pelamar kerja

- i. Pada laman Home, pilih *section* Applicant pada bagian navigation bar aplikasi
- ii. Sistem akan membawa Anda menuju laman Applicant yang mengembalikan metadata pelamar kerja hingga jumlah CV yang dibatasi
- c. Melihat semua pekerjaan yang mungkin
  - i. Pada laman Home, pilih *section* Job pada bagian navigation bar aplikasi
  - ii. Sistem akan membawa Anda menuju laman Job yang mengembalikan alamat CV pelamar kerja sesuai kategori Job yang dibatasi

#### 4. Pembagian tugas dan fase pengembangan

Terakhir, berdasarkan hasil identifikasi alur program dan fungsionalitas, dilakukan pembagian tugas sebagai berikut.

- a. Brian : Integrasi database, pengisian basis data menggunakan fake data, implementasi keseluruhan GUI, implementasi algoritma Knuth-Morris-Pratt dan Boyer-Moore, dan integrasi akhir
- b. Abrar : Penerapan enkripsi dan dekripsi, algoritma Aho Corasick, dan penerapan *fuzzy matching*
- c. Faqih : Implementasi struktur data CV dan pemrosesan CV dan bonus video

### 3.2. *Mapping* Persoalan CV ATS

Berdasarkan langkah penyelesaian yang telah dirancang, dilakukan pemetaan persoalan untuk setiap fungsionalitas. Pemetaan dilakukan dengan cara membagi fungsionalitas menjadi beberapa tanggung jawab dan melakukan *assignment* untuk setiap fungsionalitas tersebut dengan fungsi dan prosedur yang telah didefinisikan sebelumnya. Berikut pemetaan yang dilakukan.

1. GUI
  - a. Menjadi antarmuka untuk pengguna ketika memasukkan kata kunci, algoritma yang ingin digunakan, dan jumlah CV yang ingin ditampilkan

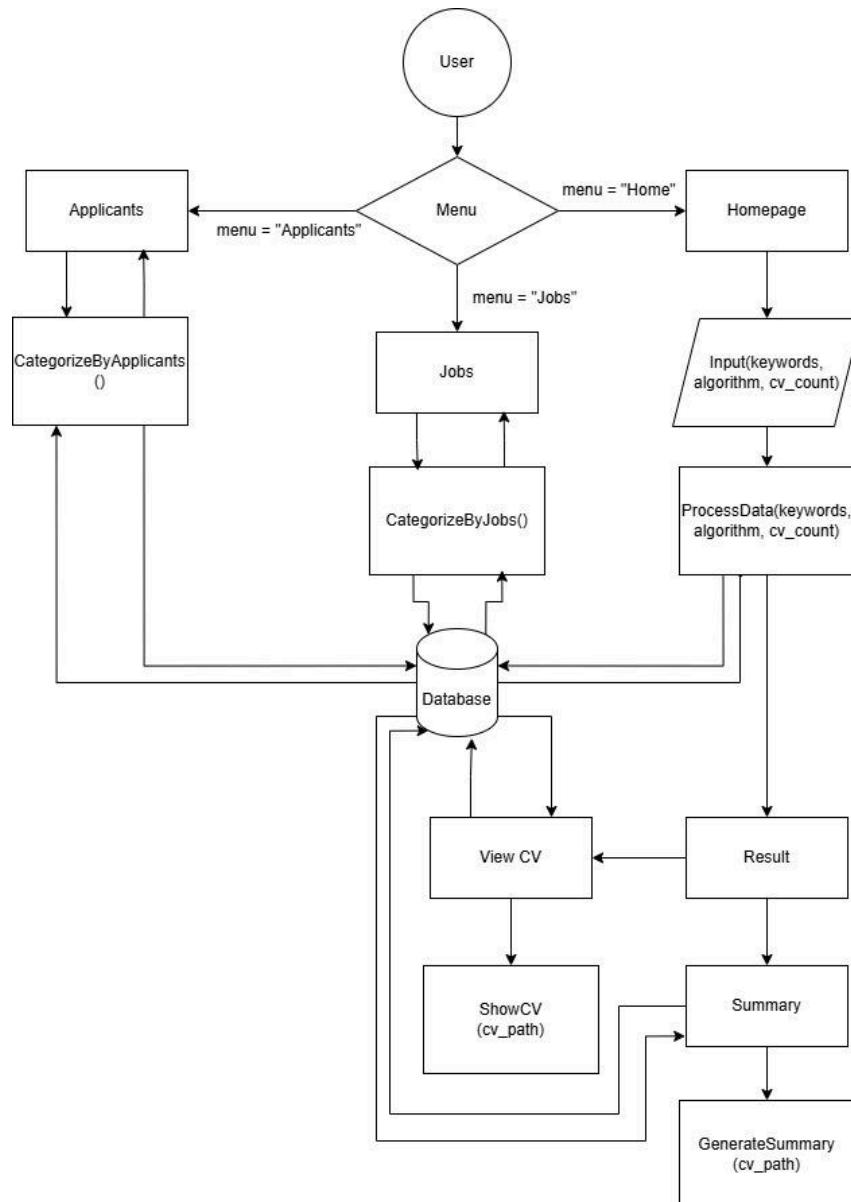
- i. Struktur data
    - 1. Page
  - ii. Fungsi dan prosedur
    - 1. start()
    - 2. encrypt()
    - 3. decrypt()
- b. Menjadi antarmuka untuk sistem ketika menampilkan hasil dari proses *pattern matching*
- i. Struktur data
    - 1. Page
    - 2. PageComponent
  - ii. Fungsi dan prosedur
    - 1. ahoCorasick()
    - 2. boyerMoore()
    - 3. knuthMorrisPratt()
    - 4. fuzzyMatch()
2. Penyimpanan metadata melalui basis data relasional
- a. Menyimpan seluruh metadata pelamar kerja dalam format nama, tanggal lahir, alamat, nomor telepon, alamat CV, dan pekerjaan yang ingin dilamar
    - i. Struktur data
      - 1. -
    - ii. Fungsi dan prosedur
      - 1. getDataFromDatabase()
3. Melakukan pencarian data terhadap kata kunci yang diberikan, baik exact matching ataupun fuzzy matching
- a. Melakukan pencarian data terhadap kata kunci yang diberikan secara exact matching melalui algoritma KMP, BM, dan AH
    - i. Struktur data
      - 1. Page
      - 2. PageComponent
    - ii. Fungsi dan prosedur

1. knuthMorrisPratt()
  2. ahoCorasick()
  3. boyerMoore()
  4. preprocessCV()
- b. Melakukan pencarian data terhadap kata kunci yang diberikan secara *fuzzy matching* melalui pendekatan *levenshtein distance*
- i. Struktur data
    1. Page
    2. PageComponent
  - ii. Fungsi dan prosedur
    1. fuzzyMatch()
4. Melakukan pemrosesan data pada CV untuk kata kunci yang diberikan untuk segmentasi CV
- a. Melakukan pemrosesan data pada CV untuk kata kunci yang tertentu untuk memperoleh rangkuman dan bagian-bagian esensial
    - i. Struktur data
      1. CV
    - ii. Fungsi dan prosedur
      1. getDataFromDatabase()
      2. preprocessCV()

### 3.3. Fungsionalitas dan Arsitektur Aplikasi

Berdasarkan pemetaan persoalan yang telah dirancang, dilakukan penyusunan desain dan komposisi terhadap arsitektur aplikasi dan fungsionalitas. Desain ini dilakukan dalam dua tahap, yakni arsitektur aplikasi yang mencakup keseluruhan komponen dan alur data dan pendaftaran fungsionalitas untuk keseluruhan aplikasi. Kedua tahap tersebut dijelaskan sebagai berikut.

1. Diagram arsitektur aplikasi



## 2. Daftar fungsionalitas untuk keseluruhan aplikasi

- Mencari pelamar kerja sesuai dengan masukan kata kunci
- Melihat semua pelamar kerja
- Melihat semua pekerjaan yang mungkin

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1. Spesifikasi Teknis Program

##### 4.1.1. Struktur Data

```

TYPE Job
    position      : STRING      { Posisi atau jabatan pekerjaan }
    range        : STRING      { Rentang waktu bekerja }
    description   : STRING      { Deskripsi singkat pekerjaan }
END TYPE

TYPE Education
    studyProgram : STRING      { Program studi atau jurusan }
    institution  : STRING      { Nama institusi pendidikan }
    rangeYear    : STRING      { Tahun atau rentang waktu studi }
END TYPE

TYPE CV
    continuousText : STRING      { Seluruh teks mentah dari file PDF }
    summary       : STRING      { Ringkasan profil dari CV }
    skills        : ARRAY OF STRING { Daftar keahlian }
    jobHistory    : ARRAY OF Job { Daftar riwayat pekerjaan, menggunakan
tipe Job }
    education     : ARRAY OF Education { Daftar riwayat pendidikan,
menggunakan tipe Education }
END TYPE

```

##### 4.1.2. Fungsi dan Prosedur

```

BEGIN PROCEDURE start(page)

    // 1. Konfigurasi Properti Halaman Utama
    SET page.title TO "CVRobin : CV Analyzer"
    SET page.scrolling TO automatic
    SET page.alignment TO center vertically and horizontally
    SET page.window_dimensions TO 1333.3 width, 1000 height
    SET page.layout TO expand and fill space

    // 2. Definisikan Prosedur untuk Menangani Perubahan Rute/URL
    BEGIN PROCEDURE handle_route_change
        // Kosongkan konten halaman saat ini
        CLEAR all views from page

        // Periksa rute yang aktif
        GET current_route from page.route

        IF current_route is "/" THEN
            ADD view_homepage TO page
    END PROCEDURE

```

```

ELSE IF current_route is "/applicants" THEN
    ADD view_applicants TO page
ELSE IF current_route is "/jobs" THEN
    ADD view_jobs TO page
END IF

// Perbarui tampilan halaman untuk menampilkan view yang baru
REFRESH page
END PROCEDURE

// 3. Hubungkan dan Jalankan
// Atur agar 'handle_route_change' dijalankan setiap kali rute berubah
ASSIGN handle_route_change TO page.on_route_change_event

// Minta halaman untuk memuat rute awal saat aplikasi pertama kali
dijalankan
NAVIGATE TO page.current_route

END PROCEDURE

BEGIN PROCEDURE setup

// 1. Inisialisasi
DEFINE data_directory, password
INITIALIZE empty lists: failed_files, all_profiles, all_applications,
generated_applicant_ids
INITIALIZE total_files_count TO 0

IF data_directory does not exist THEN
    LOG "Error: Direktori tidak ditemukan."
    EXIT PROCEDURE
END IF

// 2. Proses File dari Direktori
INITIALIZE a fake data generator

FOR EACH file in data_directory DO
    IF file is a PDF THEN
        INCREMENT total_files_count
        LOG "Memproses file: [file_path]"

    TRY
        // Generate dan enkripsi data dummy untuk satu pelamar
        CREATE a new profile_data structure
        SET profile_data.first_name TO ENCRYPT(fake_first_name, password)
        SET profile_data.last_name TO ENCRYPT(fake_last_name, password)
        // ... (lakukan hal yang sama untuk tanggal lahir, alamat, no.
telepon) ...
        ADD profile_data TO all_profiles list

        // Buat detail aplikasi
        CREATE a new application_data structure
        SET application_data.role TO ENCRYPT(fake_job_title, password)
        SET application_data.cv_path TO ENCRYPT(file_path, password)
    END TRY
END FOR

```

```

    ADD application_data TO all_applications list

    CATCH any error
        LOG "Error saat memproses file: [file_path]"
        ADD file_path TO failed_files list
    END TRY
END IF
END FOR

// 3. Tampilkan Ringkasan Hasil Proses
CALCULATE success_rate based on total_files_count and count of failed_files
LOG "Proses selesai."
LOG "Total file: [total_files_count]"
LOG "Tingkat keberhasilan: [success_rate]%"
LOG "File gagal: [count of failed_files]"

// 4. Perbarui Database
IF all_profiles list is not empty THEN
    CONNECT to the database

    // Reset tabel-tabel di database
    DROP TABLE IF EXISTS application_detail
    DROP TABLE IF EXISTS applicant_profile
    CREATE TABLE applicant_profile with schema (id, name, dob, etc.)
    CREATE TABLE application_detail with schema (id, applicant_id, role, etc.)

    // Masukkan semua data profil dan kumpulkan ID yang baru dibuat
    FOR EACH profile in all_profiles DO
        INSERT profile into applicant_profile table
        GET the last_inserted_id
        ADD last_inserted_id TO generated_applicant_ids list
    END FOR

    // Masukkan semua data detail aplikasi
    FOR EACH application in all_applications DO
        // Hubungkan detail aplikasi dengan profil secara acak
        SET random_applicant_id TO a RANDOM choice from generated_applicant_ids
list
        INSERT application (linked with random_applicant_id) into
application_detail table
    END FOR

    COMMIT database changes
    CLOSE database connection
END IF

END PROCEDURE

BEGIN PROCEDURE ahoCorasickMatch(text, list_of_keywords)

    // Langkah 1: Bangun State Aho-Corasick
    // Ini adalah langkah pra-pemrosesan yang paling penting.
    // Sebuah 'trie' (pohon prefix) dibuat dari semua keyword, kemudian
    // 'failure links' (link kegagalan) ditambahkan ke setiap node.

```

```

SET machine TO the result of buildAhoCorasickMachine(list_of_keywords).

// Langkah 2: Inisialisasi Pencarian
INITIALIZE an empty map named 'results' to store keyword counts.
SET current_state TO the root node of the machine (state 0).

// Langkah 3: Proses Teks per Karakter (Single Pass)
FOR EACH character IN the lowercase version of the text DO

    // A. Ikuti 'Failure Links' jika transisi normal tidak ada.
    // Selama tidak ada 'child' untuk karakter saat ini DAN kita tidak di
root,
        // lompat ke state kegagalan. Ini seperti bertanya, "Adakah awalan
(prefix)
        // lain yang lebih pendek yang cocok dengan akhir dari apa yang baru saja
kita lihat?"
        WHILE the current_state has no transition for the character AND
current_state is not the root node DO
            SET current_state TO the 'fail' link of the current_state.
        END WHILE

    // B. Lakukan Transisi Normal jika memungkinkan.
    // Jika ada 'child' untuk karakter saat ini dari state (setelah mungkin
melompat),
        // pindah ke state tersebut.
        IF the current_state has a transition for the character THEN
            SET current_state TO the child node for that character.
        END IF

    // C. Periksa dan Catat Output.
    // Periksa apakah state kita saat ini menandai akhir dari satu atau lebih
keyword.
        IF the 'output' list of the current_state is not empty THEN
            // Jika ya, kita telah menemukan satu atau lebih kecocokan.
            FOR EACH found_keyword in the 'output' list of the current_state DO
                INCREMENT the count for found_keyword in the 'results' map.
            END FOR
        END IF

    END FOR

// Langkah 4: Kembalikan Hasil
RETURN the 'results' map.

END PROCEDURE
BEGIN PROCEDURE boyerMooreMatch(text, list_of_keywords)

    // 1. Inisialisasi dan Pra-pemrosesan Teks
    INITIALIZE an empty map named 'results' to store keyword counts.
    PREPROCESS the input text (e.g., remove non-alphanumeric characters) into
'clean_text'.

    // 2. Lakukan pencarian untuk setiap keyword
    FOR EACH keyword IN list_of_keywords DO

```

```

// Lanjutkan hanya jika keyword bisa muat di dalam teks
IF length of clean_text is greater than or equal to length of keyword THEN

    // Langkah A: Pra-pemrosesan Boyer-Moore (Membuat Tabel Karakter Buruk)
    // Tabel ini menyimpan posisi kemunculan terakhir setiap karakter di
dalam keyword.
    // Ini adalah kunci untuk "lompatan cerdas".
    SET bad_character_table TO the result of getBadMatchTable(keyword).

    // Langkah B: Inisialisasi Pointer
    // Pointer disejajarkan dengan akhir dari kemungkinan kecocokan pertama.
INITIALIZE text_pointer TO (length of keyword - 1).

    // Langkah C: Loop Pencocokan Utama (berjalan selama pointer teks masih
di dalam batas)
    WHILE text_pointer is less than the length of clean_text DO
        INITIALIZE keyword_pointer TO (length of keyword - 1).
        SET starting_text_pos_for_this_alignment TO text_pointer.

        // Loop internal virtual (membandingkan dari kanan ke kiri)
        // Kasus 1: Karakter cocok DAN seluruh keyword telah terverifikasi
        IF clean_text[text_pointer] EQUALS keyword[keyword_pointer] AND
keyword_pointer EQUALS 0 THEN
            // KECOCOKAN DITEMUKAN!
            INCREMENT the count for the keyword in the 'results' map.
            // Lompatkan teks sejauh panjang keyword untuk mencari kecocokan
berikutnya
            SET text_pointer TO starting_text_pos_for_this_alignment + length of
keyword.
            RESET keyword_pointer TO the end of the keyword.

        // Kasus 2: Karakter cocok, tetapi masih ada sisa keyword untuk
diperiksa
        ELSE IF clean_text[text_pointer] EQUALS keyword[keyword_pointer] THEN
            // Mundur satu langkah untuk melanjutkan perbandingan kanan-ke-kiri
            DECREMENT text_pointer.
            DECREMENT keyword_pointer.

        // Kasus 3: Terjadi ketidakcocokan (Mismatch)
        ELSE
            // Dapatkan karakter pada teks yang menyebabkan ketidakcocokan
            SET mismatched_char TO
clean_text[starting_text_pos_for_this_alignment].

            // Gunakan Tabel Karakter Buruk untuk menentukan seberapa jauh harus
melompat
            SET shift_amount TO 0.
            IF mismatched_char exists in the bad_character_table THEN
                SET last_occurrence TO bad_character_table[mismatched_char].
                IF last_occurrence is to the left of the current keyword_pointer
THEN
                    // Geser untuk menyejajarkan karakter yang cocok
                    SET shift_amount TO keyword_pointer - last_occurrence.

```

```

        ELSE
            // Pergeseran aman jika karakter muncul di sebelah kanan
            titik mismatch
                SET shift_amount TO 1. // Atau pergeseran aman lainnya
                END IF
            ELSE
                // Jika karakter tidak ada sama sekali di keyword, kita bisa
                melompat paling jauh
                    SET shift_amount TO length of keyword.
                END IF

                // Terapkan lompatan dan reset pointer
                SET text_pointer TO starting_text_pos_for_this_alignment + max(1,
shift_amount).
                RESET keyword_pointer TO the end of the keyword.
            END IF
        END WHILE
    END IF
END FOR

// 3. Kembalikan Hasil
RETURN the 'results' map.

END PROCEDURE

BEGIN PROCEDURE knuthMorrisPrattMatch(text, list_of_keywords)

// 1. Inisialisasi
INITIALIZE an empty map named 'results' to store the count of each keyword.

// 2. Lakukan pencarian untuk setiap keyword
FOR EACH keyword IN list_of_keywords DO

    // Langkah A: Pra-pemrosesan KMP (Membuat Tabel Batas/LPS)
    // Tabel ini berisi informasi untuk "lompatan cerdas" saat terjadi
    ketidakcocokan.
    SET border_table TO the result of getBorderTable(keyword)

    // Langkah B: Inisialisasi Pointer
    INITIALIZE text_pointer TO 0
    INITIALIZE keyword_pointer TO 0

    // Langkah C: Loop Pencocokan Utama
    WHILE text_pointer is less than the length of the text DO

        // Kasus 1: Karakter cocok
        IF text[text_pointer] EQUALS keyword[keyword_pointer] THEN
            // Periksa apakah seluruh keyword sudah cocok
            IF keyword_pointer is at the last index of the keyword THEN
                // KECOCOKAN DITEMUKAN!
                INCREMENT the count for the keyword in the 'results' map.
                // Reset pointer keyword untuk mencari kecocokan berikutnya
                SET keyword_pointer TO 0
            ELSE

```

```

        // Kecocokan parsial, lanjutkan ke karakter berikutnya
        INCREMENT text_pointer
        INCREMENT keyword_pointer
    END IF

    // Kasus 2: Karakter tidak cocok
    ELSE
        // Periksa apakah ada kecocokan parsial sebelumnya
        IF keyword_pointer > 0 THEN
            // Ini adalah inti dari KMP: Lompatkan keyword_pointer
            // ke posisi yang benar berdasarkan border_table, tanpa menggeser
text_pointer.
            SET keyword_pointer TO the value in border_table[keyword_pointer - 1]
        ELSE
            // Tidak ada kecocokan sama sekali, geser text_pointer
            INCREMENT text_pointer
        END IF
    END IF

    END WHILE
END FOR

// 3. Kembalikan Hasil
RETURN the 'results' map

END PROCEDURE

BEGIN PROCEDURE fuzzyMatch(list_of_keywords, search_text,
max_distance_threshold)

    // 1. Persiapan Teks dan Inisialisasi
    CONVERT search_text to lowercase and SPLIT into a list named words_in_text
    INITIALIZE match_count to 0

    // 2. Iterasi untuk Membandingkan Setiap Kata
    FOR EACH word in words_in_text DO
        FOR EACH keyword in list_of_keywords DO

            // 3. Hitung Jarak Levenshtein
            // Jarak Levenshtein mengukur "seberapa berbeda" dua kata.
            // Jarak 0 berarti identik. Jarak 1 berarti butuh 1 perubahan
            // (sisip/hapus/ganti).
            CALCULATE the Levenshtein_distance between the lowercase version of the
            keyword and the current word

            // 4. Periksa Apakah Cocok
            IF Levenshtein_distance is less than or equal to max_distance_threshold
            THEN
                // Jika jaraknya cukup dekat, anggap sebagai kecocokan
                INCREMENT match_count by 1
                LOG "Ditemukan kecocokan antara 'word' dan 'keyword' dengan jarak:
                [Levenshtein_distance]"
            END IF

```

```

        END FOR
    END FOR

    // 5. Kembalikan Hasil
    RETURN match_count

END PROCEDURE

BEGIN PROCEDURE levenshtein_distance(string1, string2)

    // 1. Inisialisasi
    SET len1 TO the length of string1
    SET len2 TO the length of string2
    CREATE a 2D matrix of size (len1 + 1) x (len2 + 1)

    // 2. Isi baris dan kolom pertama (biaya dasar)
    // Biaya mengubah string kosong menjadi string lain adalah panjang string
    tersebut (hanya operasi sisip).
    FOR i FROM 0 TO len1 DO
        SET matrix[i][0] TO i
    END FOR

    FOR j FROM 0 TO len2 DO
        SET matrix[0][j] TO j
    END FOR

    // 3. Hitung biaya untuk sisa matriks
    // Iterasi melalui setiap karakter dari kedua string.
    FOR i FROM 1 TO len1 DO
        FOR j FROM 1 TO len2 DO

            // Tentukan biaya substitusi
            IF string1[i-1] EQUALS string2[j-1] THEN
                SET substitution_cost TO 0 // Tidak ada biaya jika karakter sama
            ELSE
                SET substitution_cost TO 1 // Biaya 1 jika karakter berbeda
            END IF

            // Hitung biaya dari tiga kemungkinan operasi:
            SET deletion_cost TO matrix[i-1][j] + 1
            SET insertion_cost TO matrix[i][j-1] + 1
            SET match_or_substitute_cost TO matrix[i-1][j-1] + substitution_cost

            // Pilih biaya termurah dari ketiga kemungkinan
            SET matrix[i][j] TO the MINIMUM of (deletion_cost, insertion_cost,
            match_or_substitute_cost)

        END FOR
    END FOR

    // 4. Kembalikan hasil akhir
    // Hasilnya adalah nilai di sudut kanan bawah matriks.
    RETURN matrix[len1][len2]

```

```

END PROCEDURE

BEGIN PROCEDURE decrypt(password, encryptedData)

    // 1. Memisahkan data terenkripsi dan tag otentikasi
    TRY
        SET ciphertext TO all bytes of encryptedData except the last 16
        SET receivedTag TO the last 16 bytes of encryptedData
    CATCH error
        THROW "Error: Paket terenkripsi tidak valid atau korup."
    END TRY

    // 2. Menghasilkan kembali material kunci dari password
    SET keyMaterial TO generate_key_from_password(password, salt, length=48)
    SET lfsrSeedBytes TO the first 16 bytes of keyMaterial
    SET hmacKey TO the remaining bytes of keyMaterial

    // 3. Verifikasi integritas dan otentisitas data
    SET calculatedTag TO calculate_hmac_tag(ciphertext, hmacKey)

    IF calculatedTag IS NOT EQUAL TO receivedTag THEN
        THROW

```

## 4.2. Penjelasan Tata Cara Penggunaan Program

### 4.2.1. Mencari pelamar kerja sesuai dengan masukan kata kunci

1. Pada laman Home, ketik kata kunci pada kolom kata kunci dan tekan 'Enter' setiap selesai memasukkan kata kunci
2. Pilih algoritma yang ingin digunakan untuk mencari kata kunci tersebut
3. Pilih jumlah CV yang ingin ditampilkan
4. Sistem akan membawa Anda menuju laman Hasil yang mengembalikan metadata pelamar kerja serta jumlah ditemukannya kata kunci untuk setiap pelamar kerja

### 4.2.2. Melihat semua pelamar kerja

1. Pada laman Home, pilih *section* Applicant pada bagian navigation bar aplikasi
2. Sistem akan membawa Anda menuju laman Applicant yang mengembalikan metadata pelamar kerja hingga jumlah CV yang dibatasi

### 4.2.3. Melihat semua pekerjaan yang mungkin

1. Pada laman Home, pilih *section* Job pada bagian navigation bar aplikasi
2. Sistem akan membawa Anda menuju laman Job yang mengembalikan alamat CV pelamar kerja sesuai kategori Job yang dibatasi

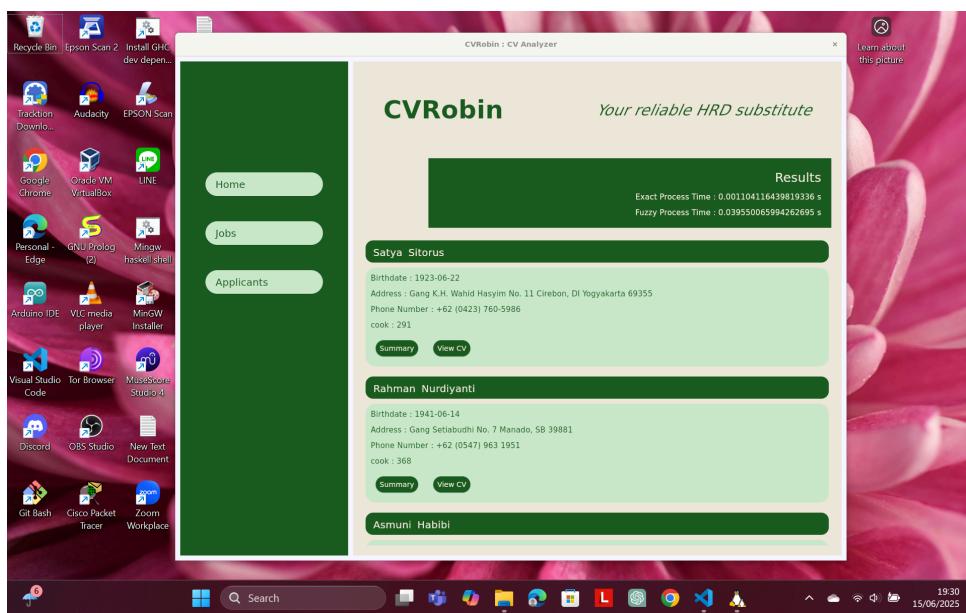
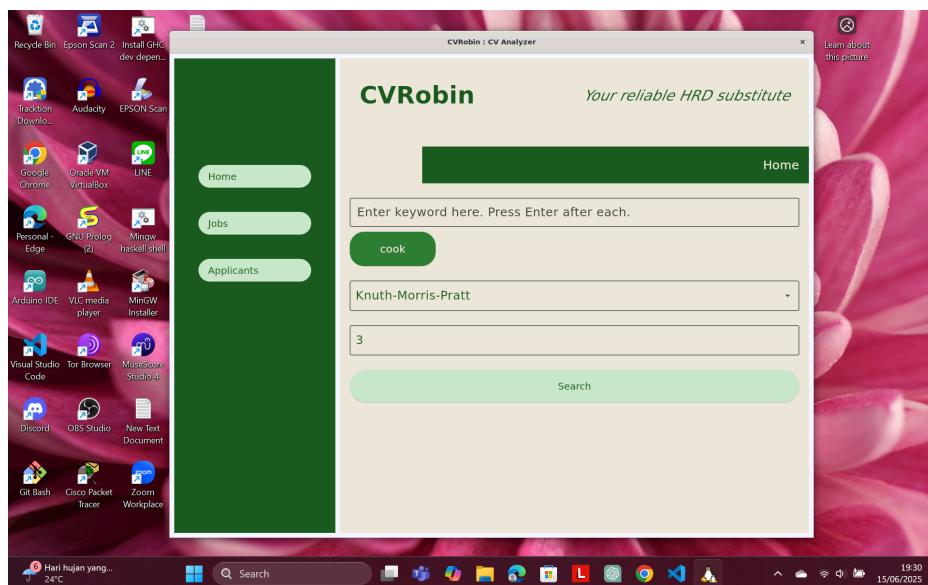
### 4.3. Pengujian

Dalam pengujian ini, dilakukan pengujian dengan empat kasus utama, yakni :

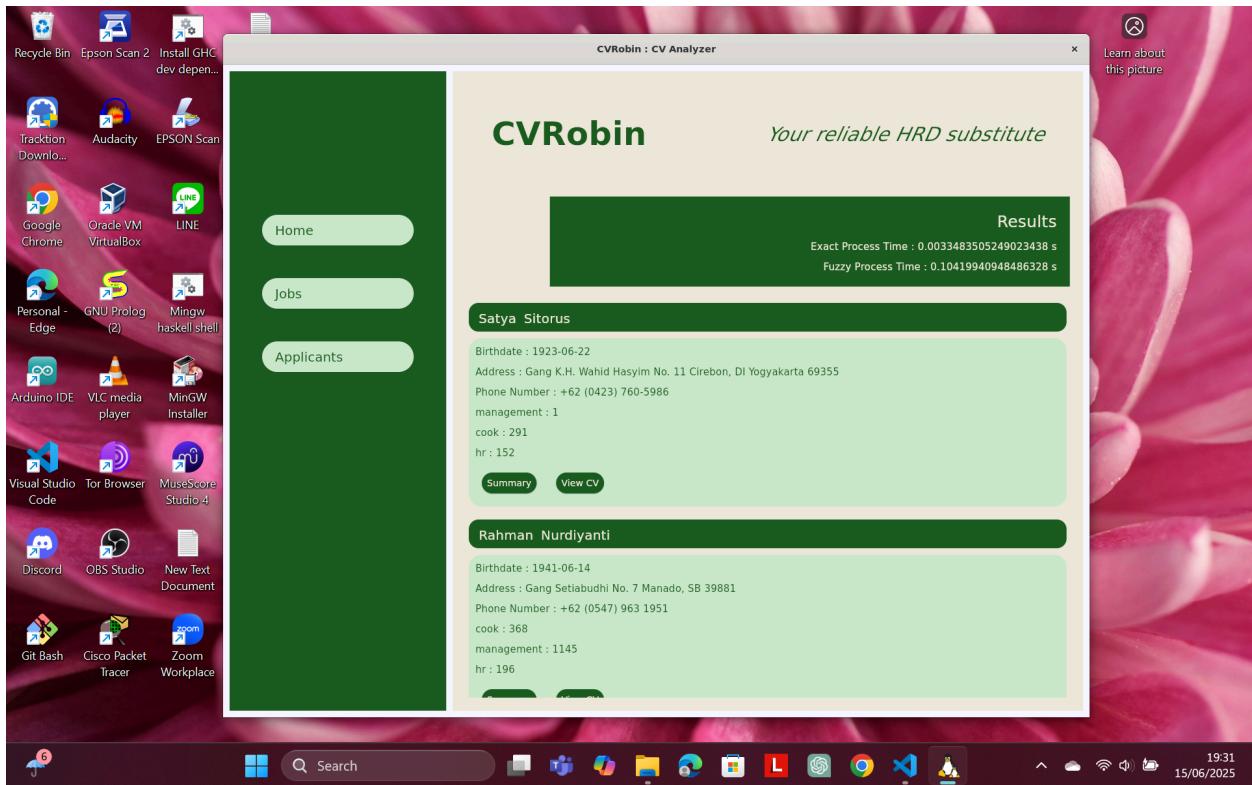
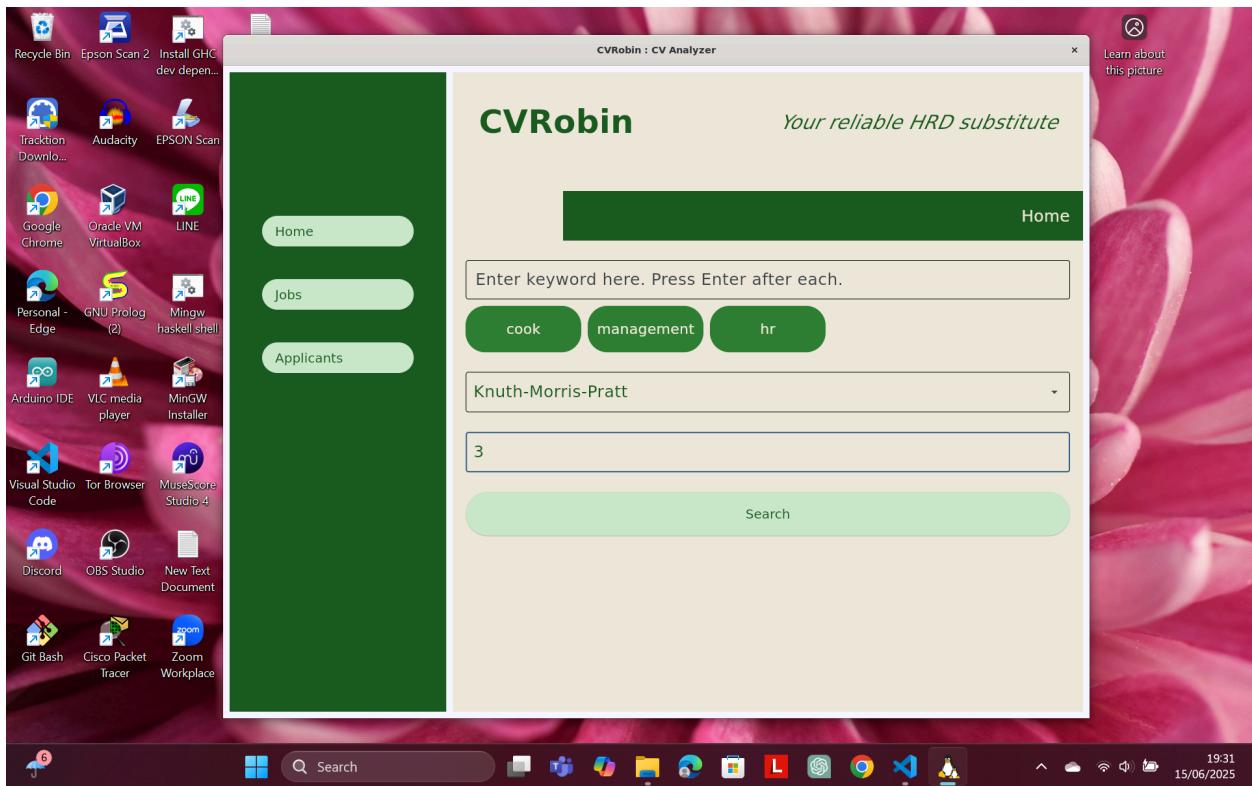
1. Test case 1 : Satu kata kunci, tidak ada kata typo
2. Test case 2 : Lebih dari satu kata kunci, tidak ada kata typo
3. Test case 3: Satu kata kunci, satu kata typo
4. Test case 4 : Lebih dari satu kata kunci, satu atau lebih dari satu kata typo

#### 4.3.1 Knuth-Morris-Pratt

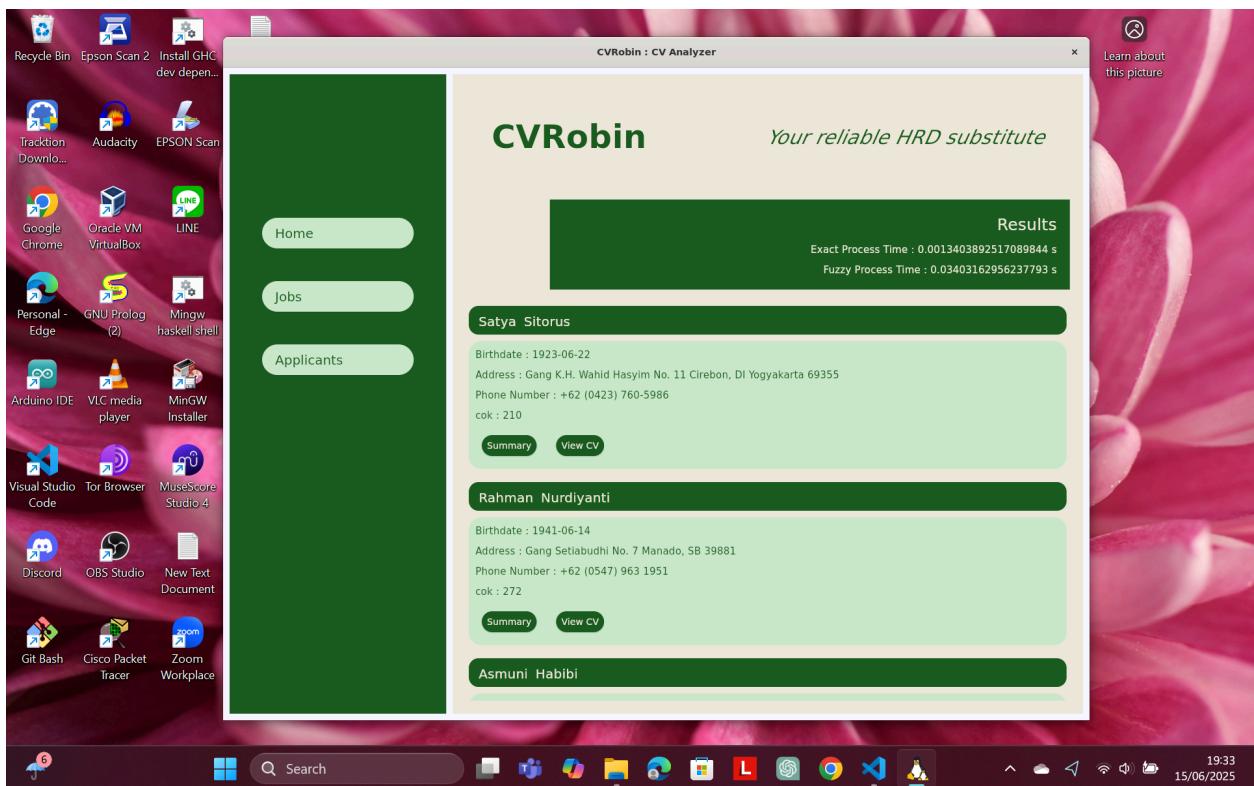
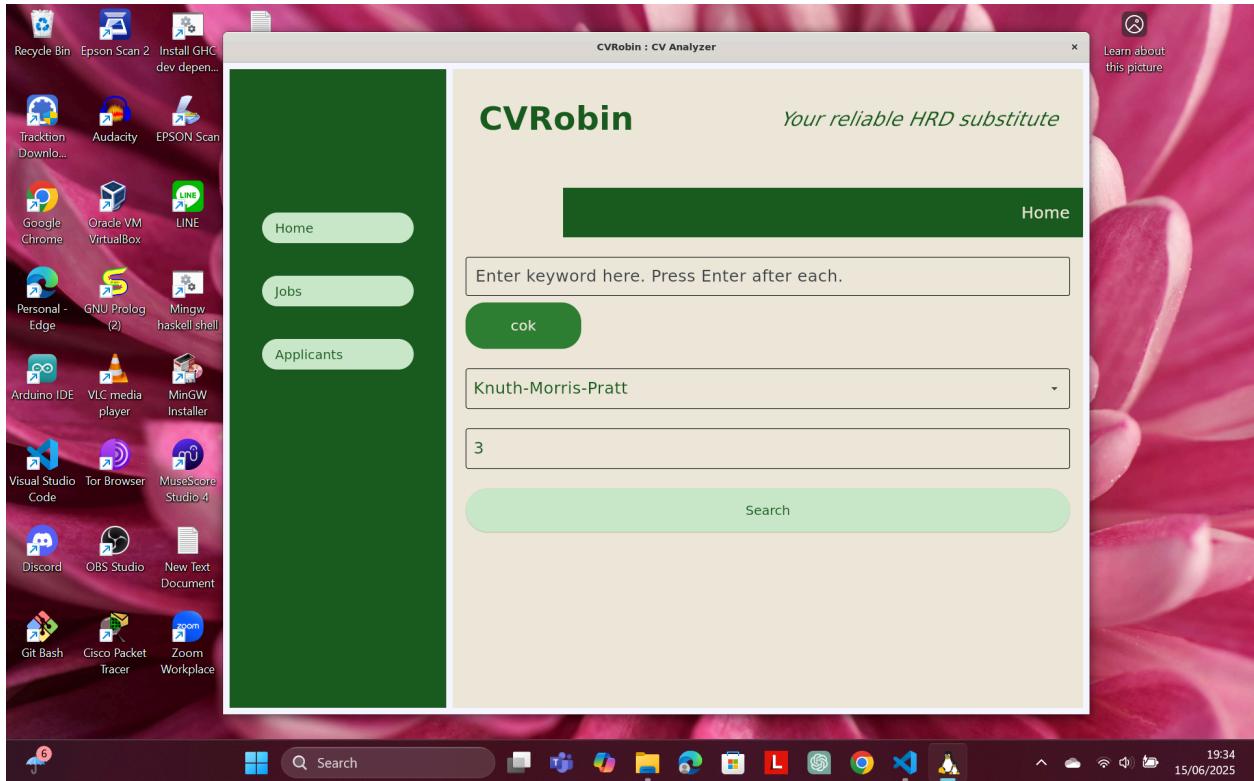
1. Test case 1



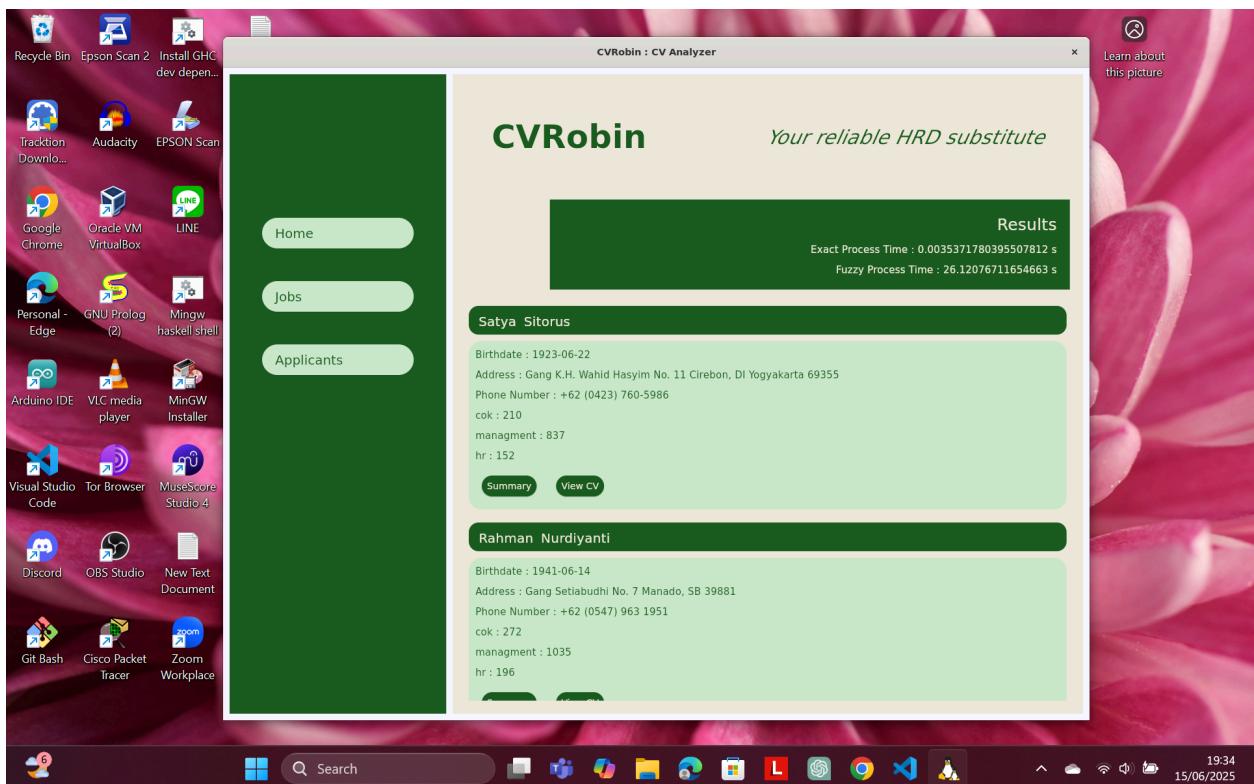
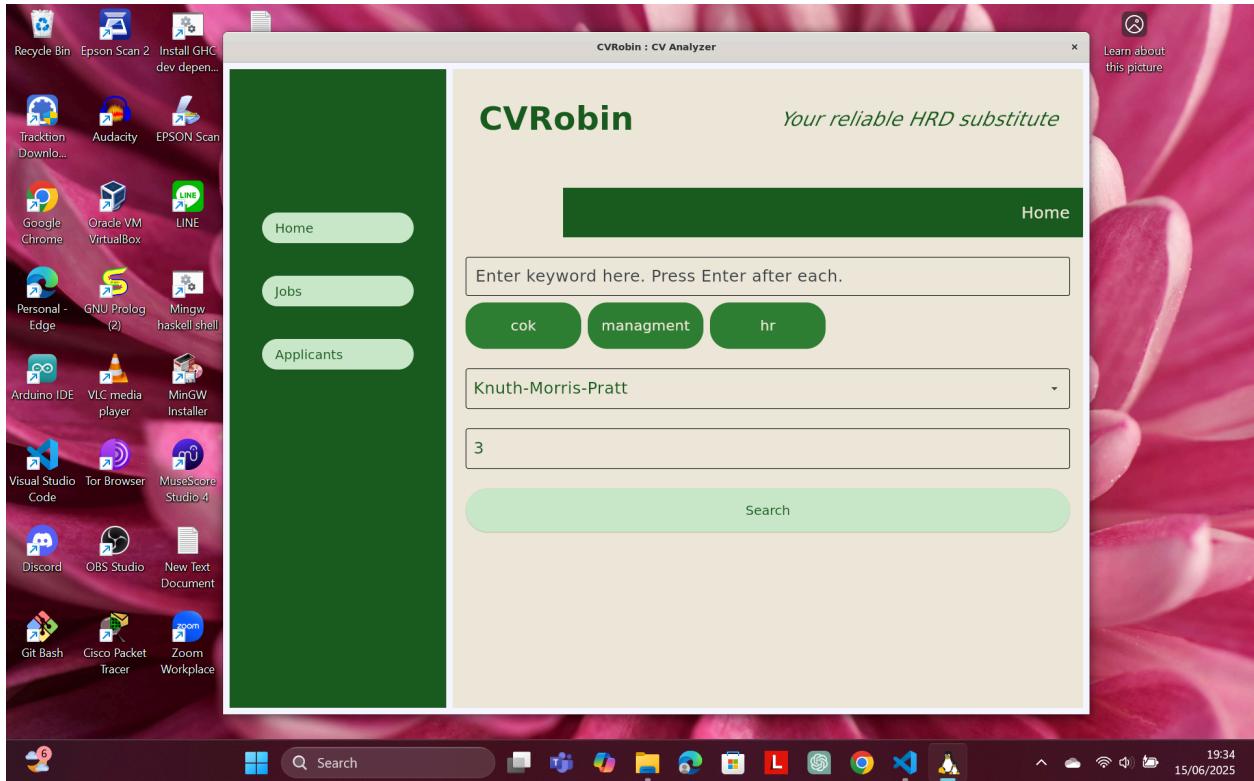
## 2. Test case 2



## 3. Test case 3

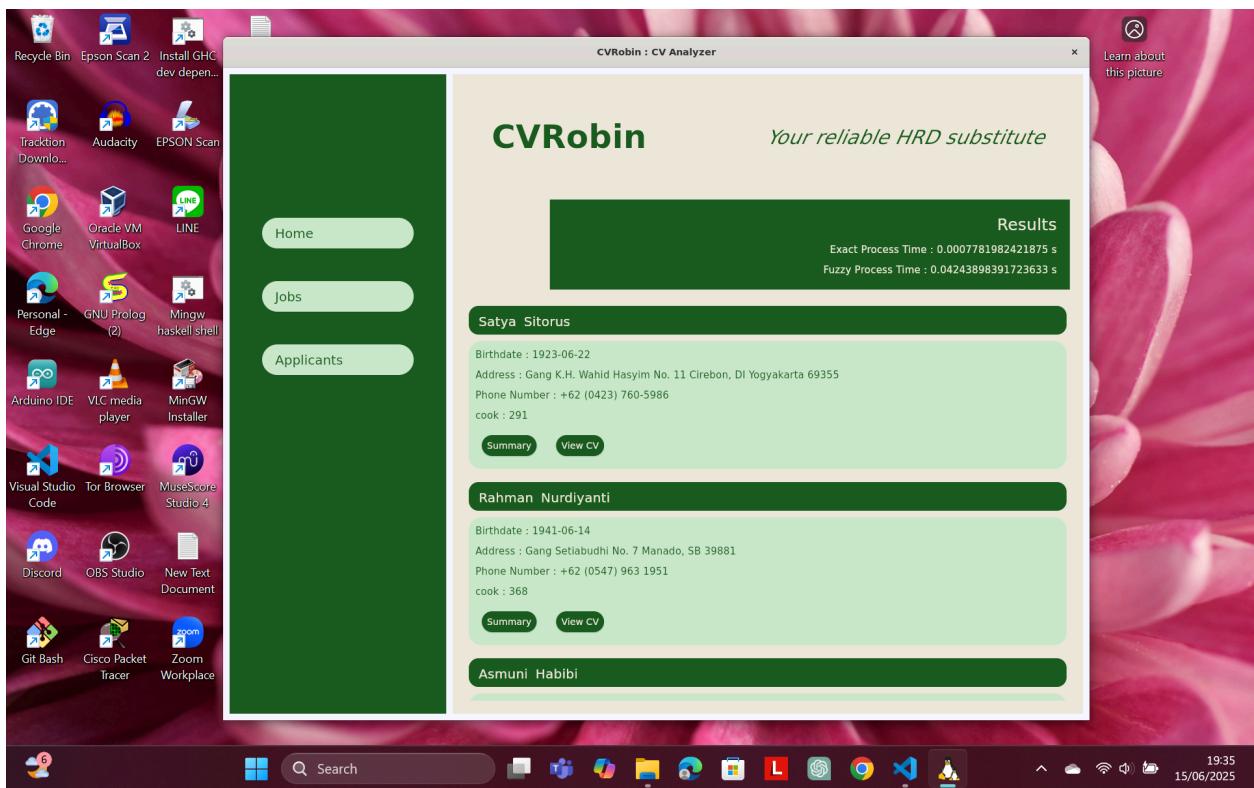
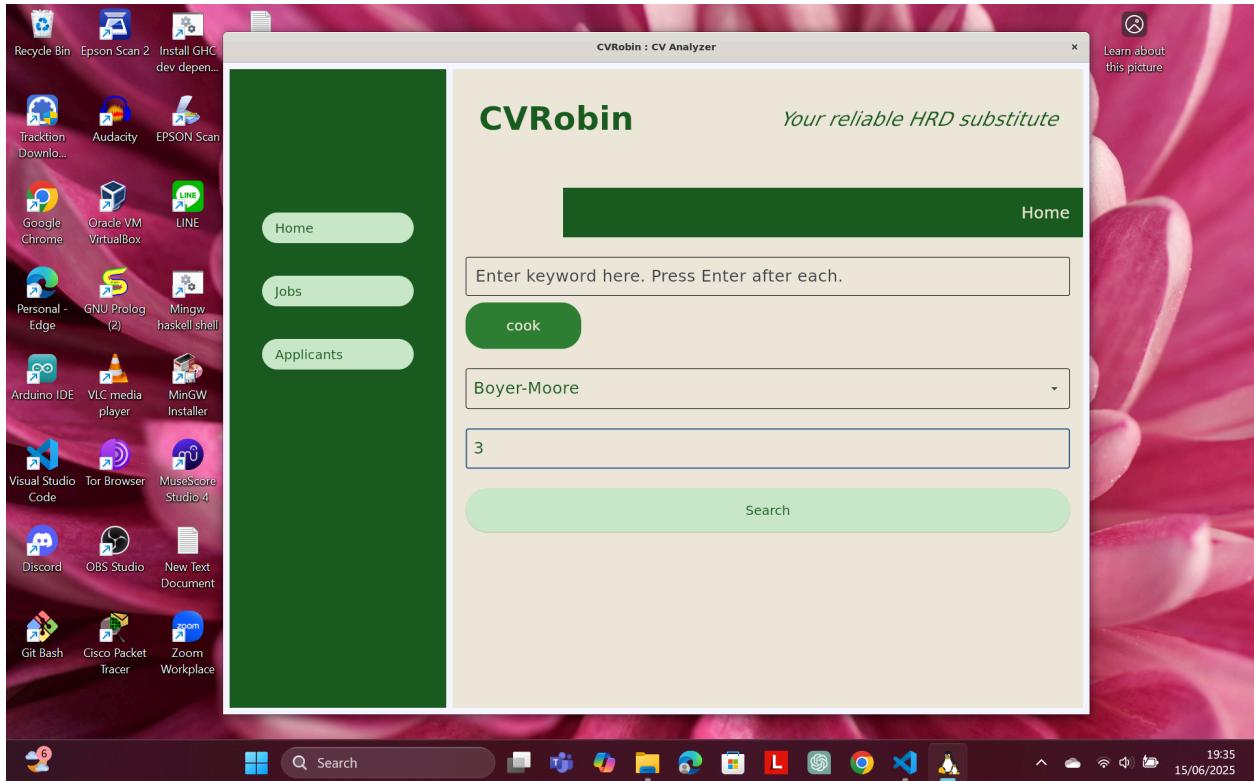


#### 4. Test case 4

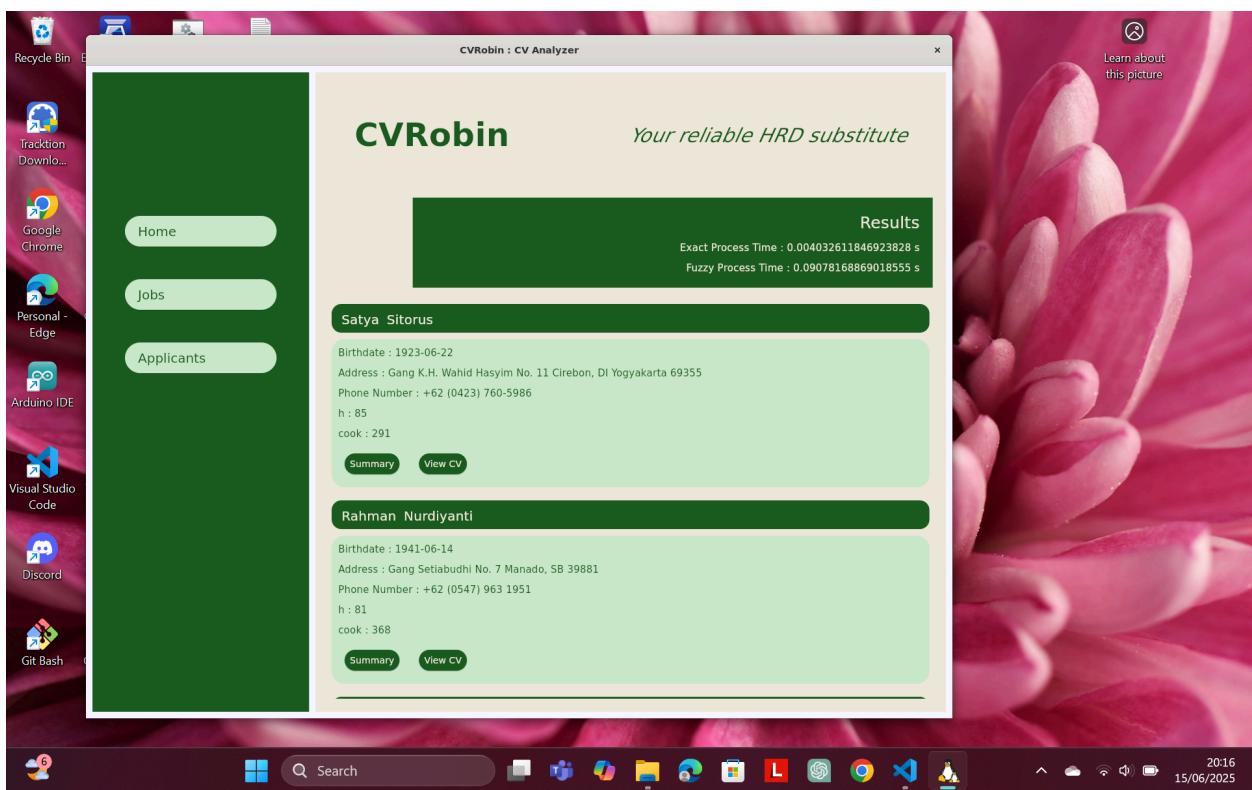
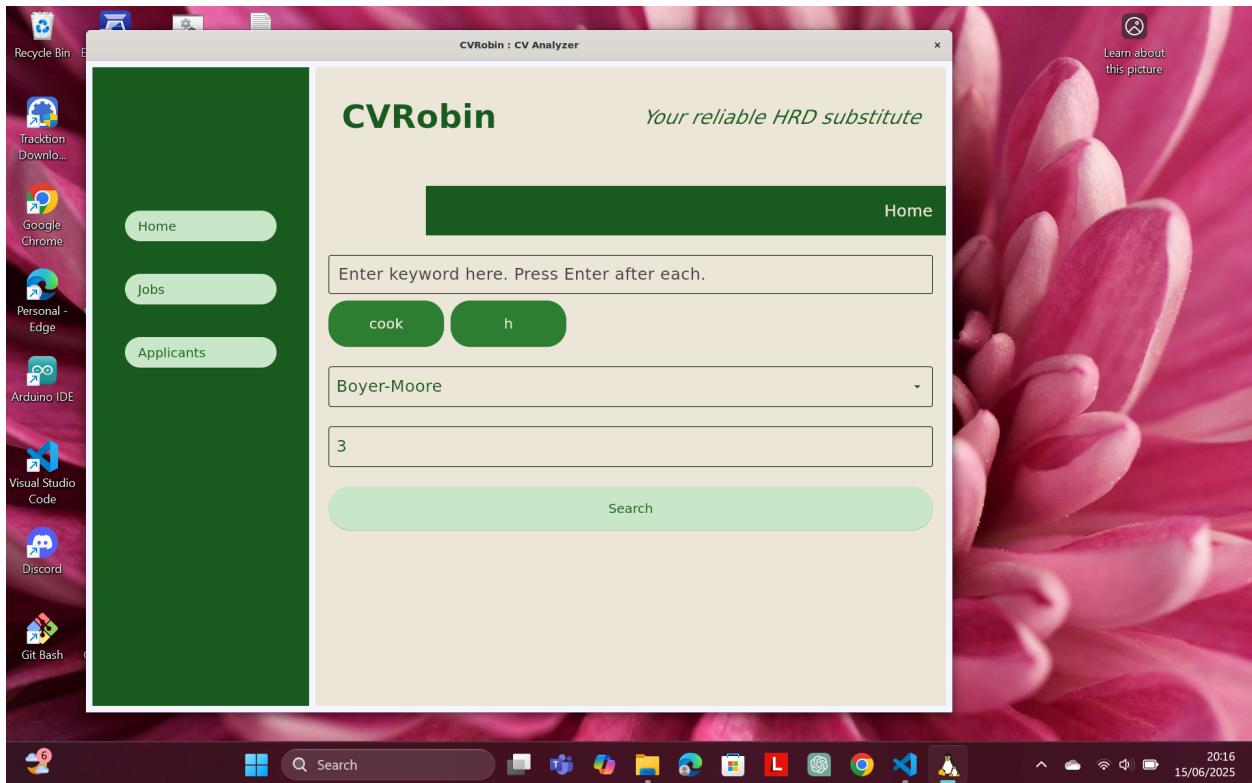


### 4.3.2 Boyer-Moore

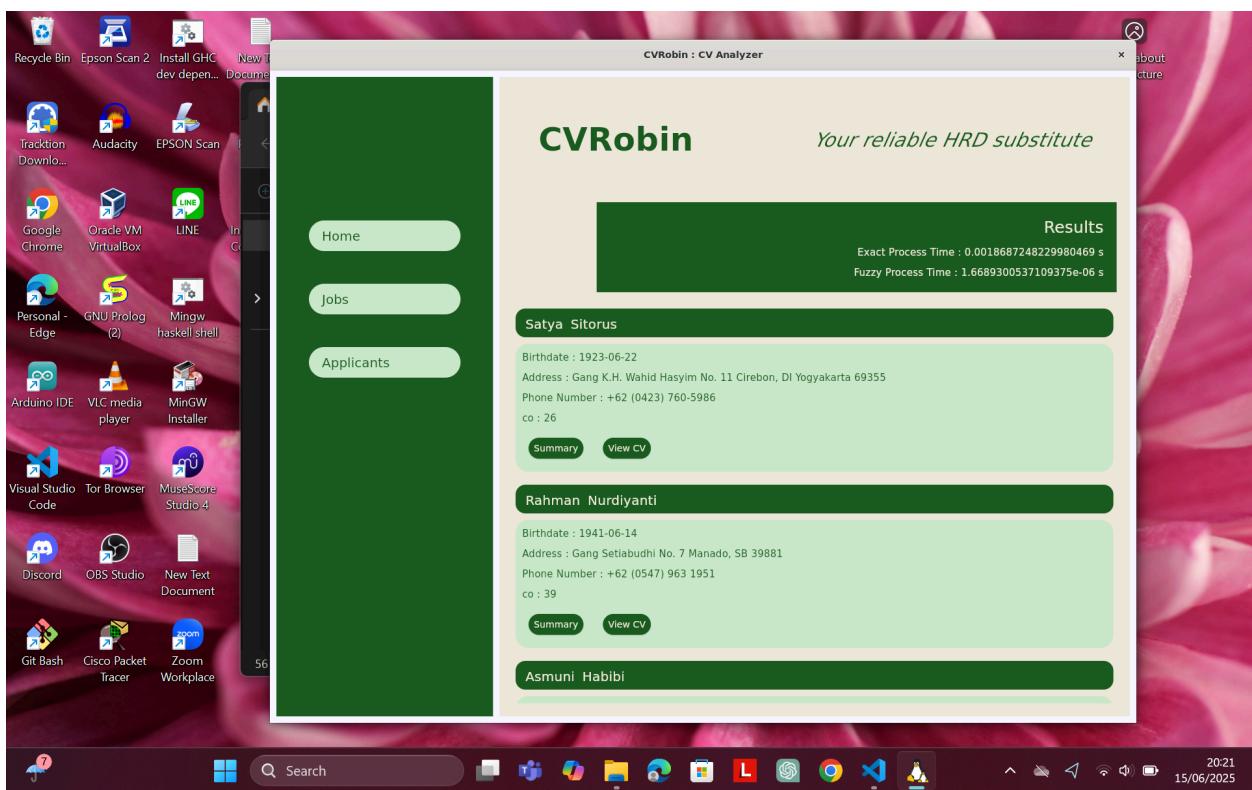
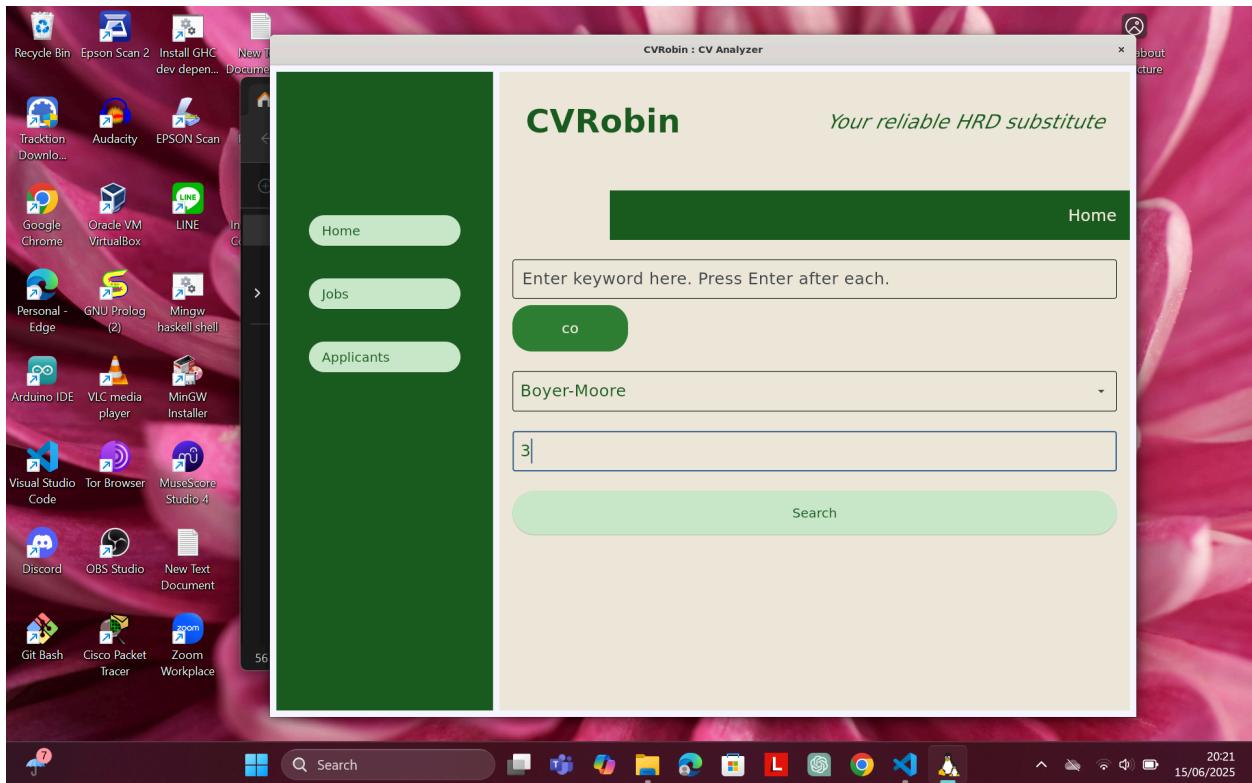
#### 1. Test case 1



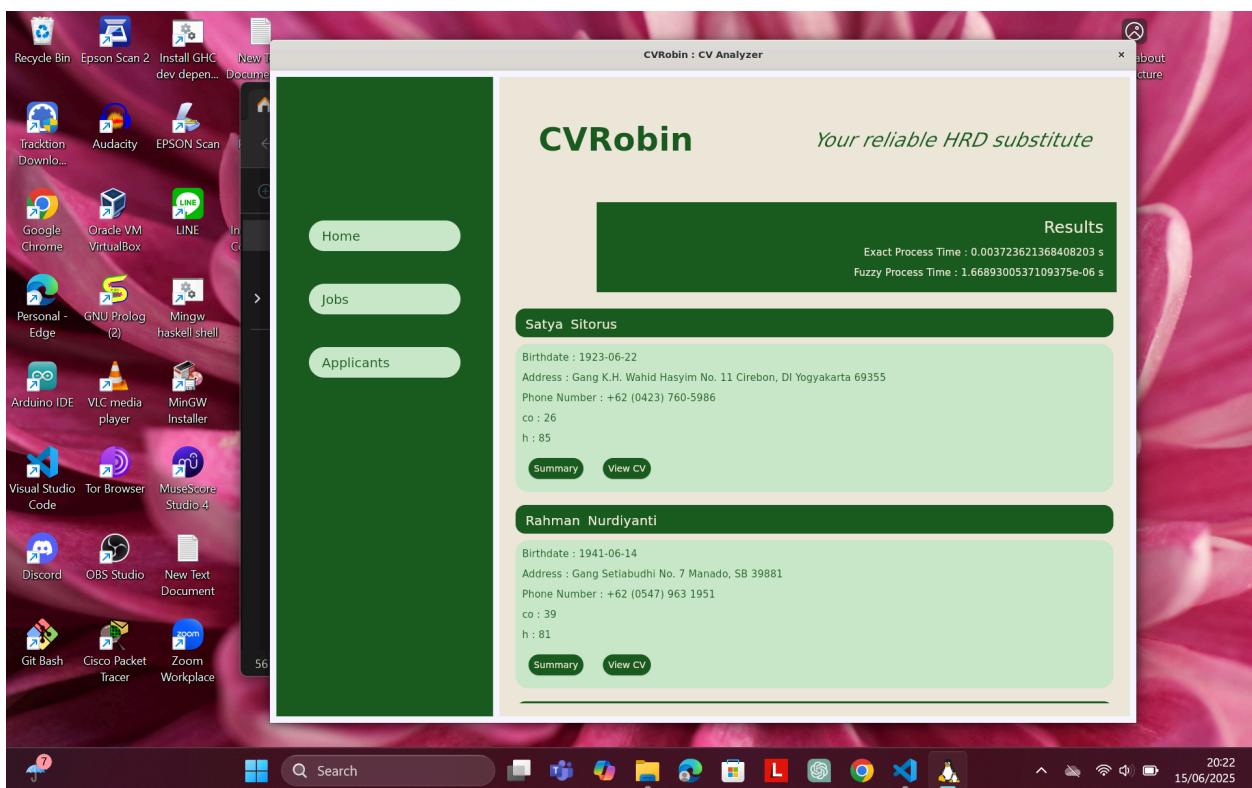
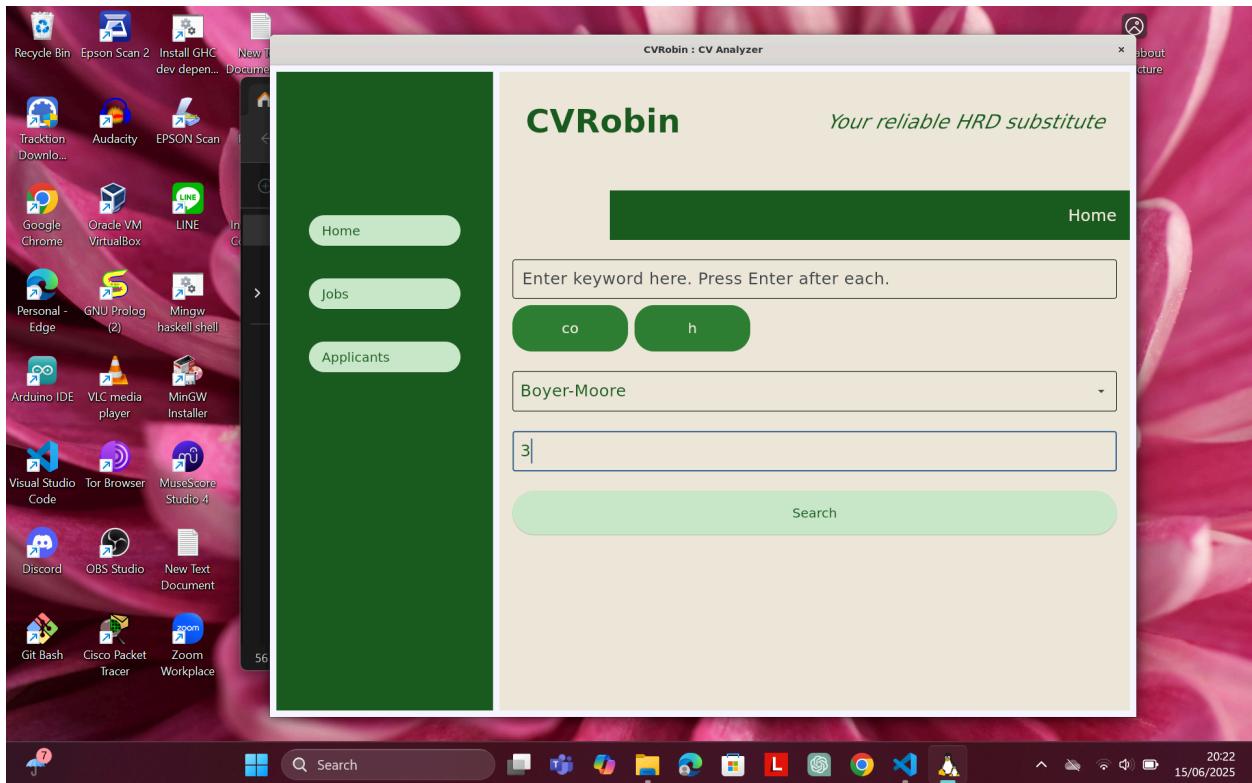
## 2. Test case 2



### 3. Test case 3

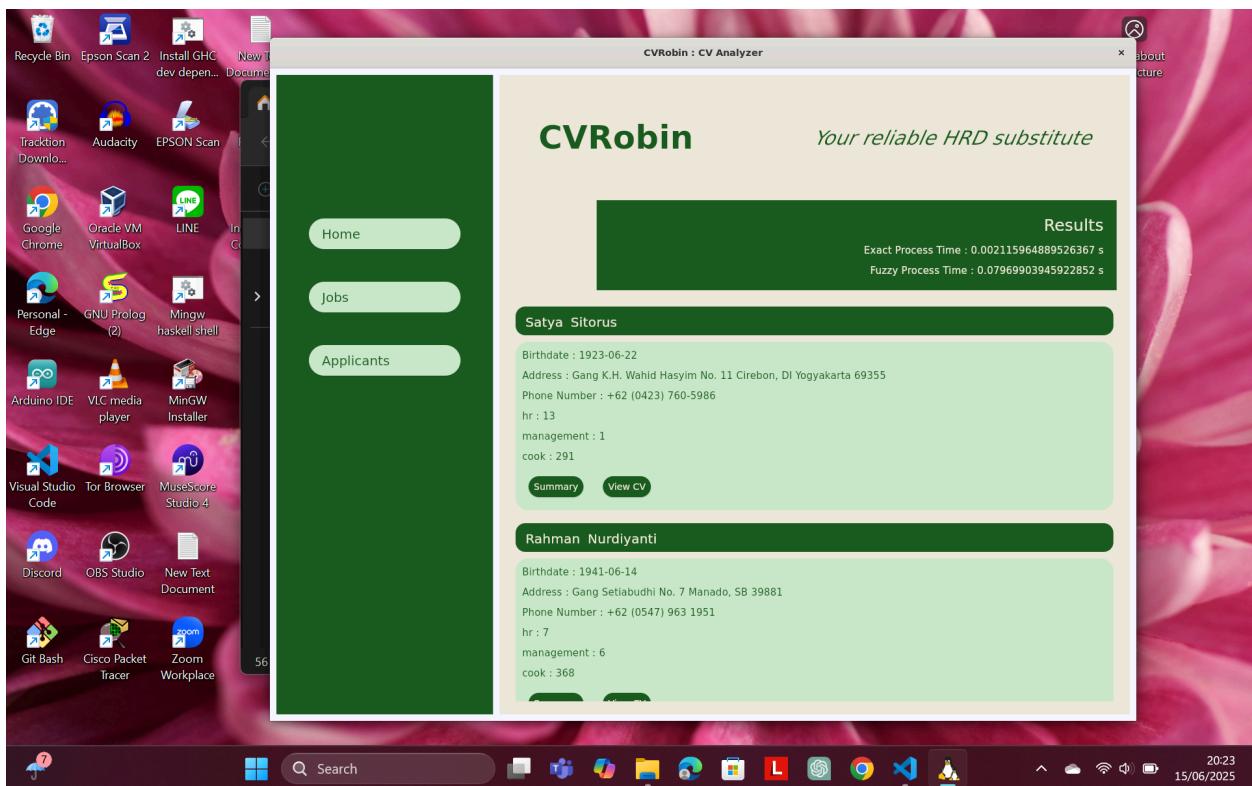
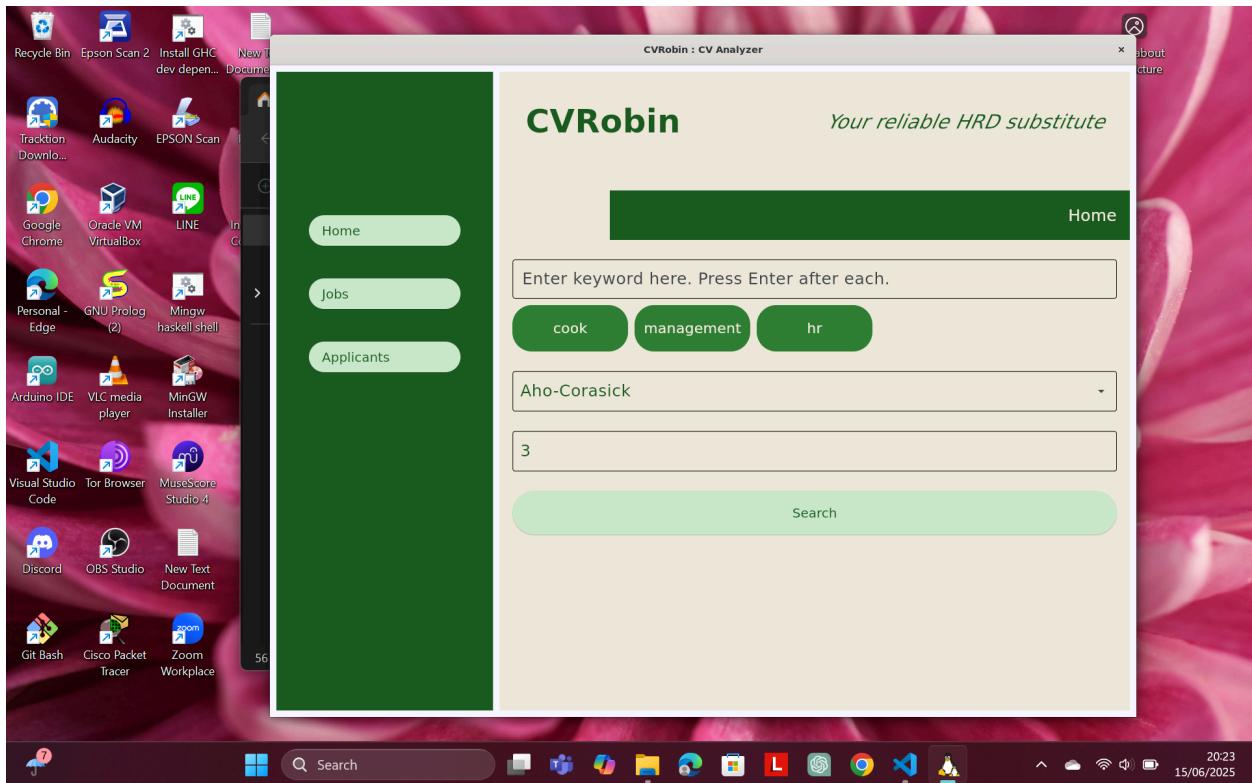


#### 4. Test case 4

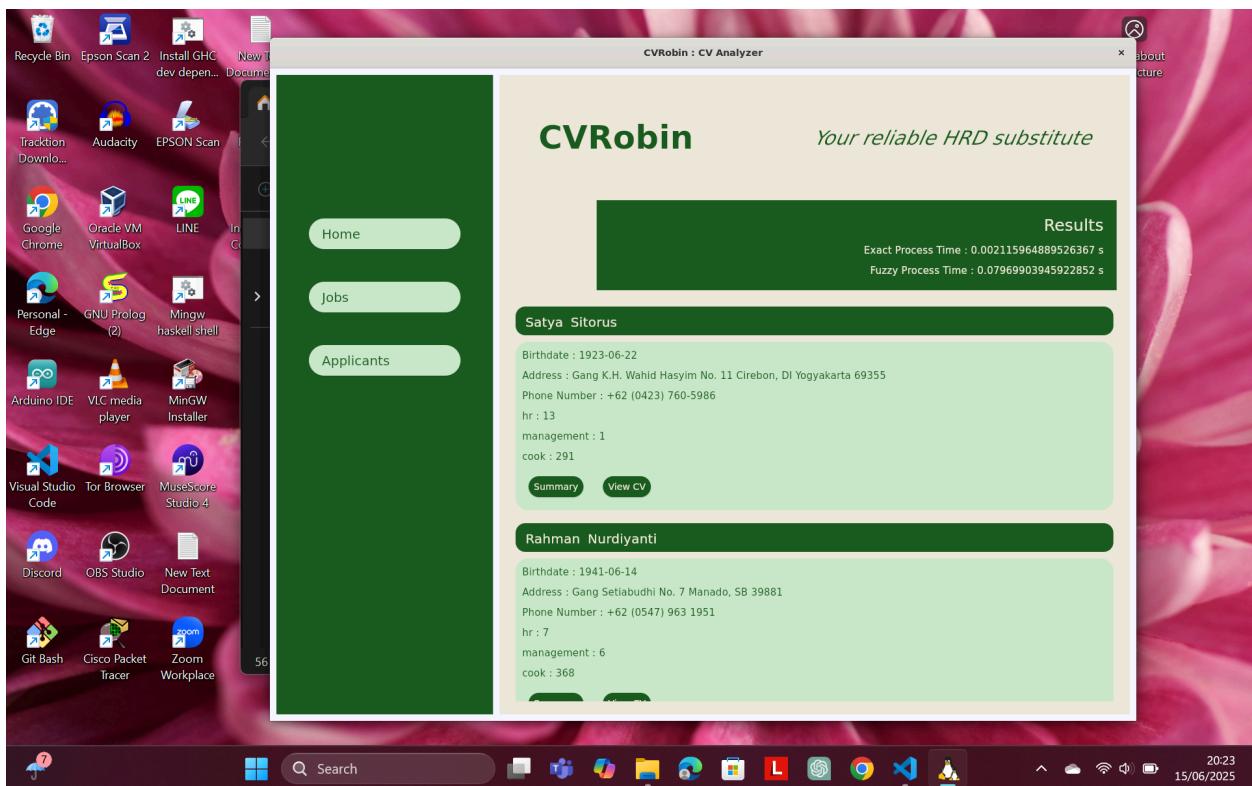
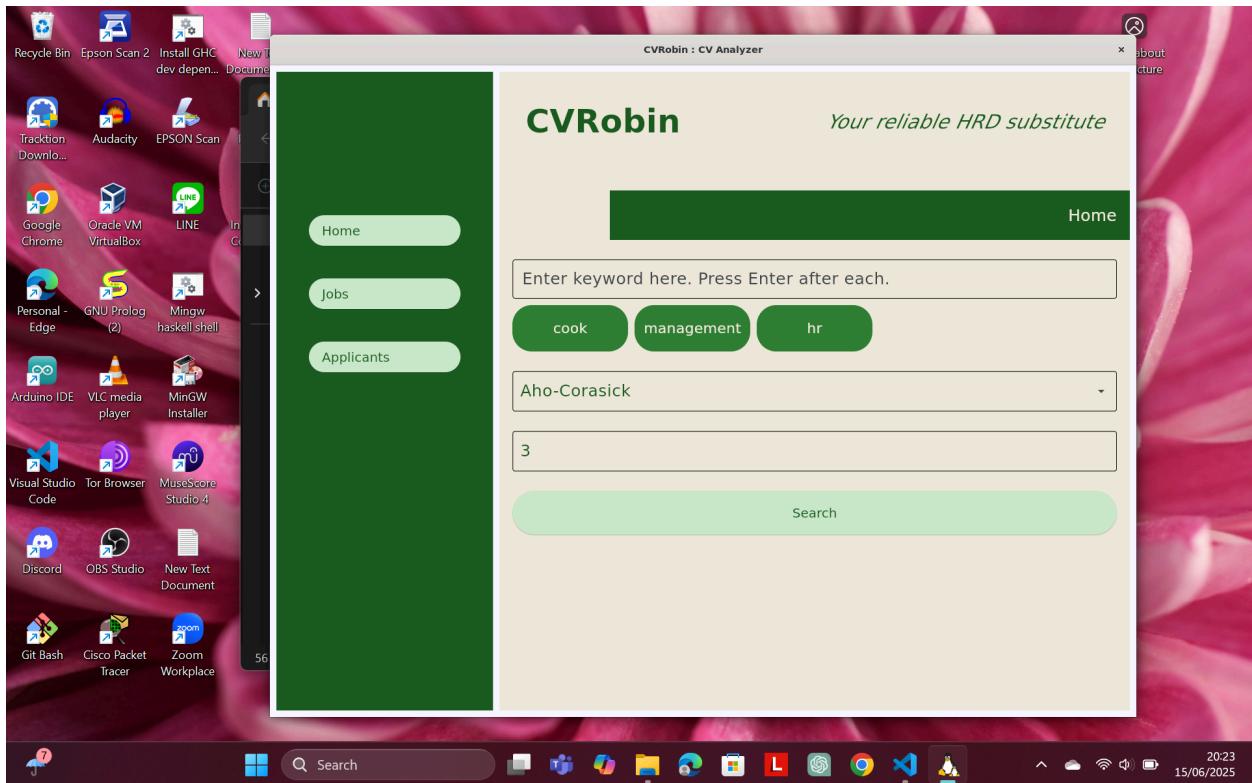


### 4.3.3 Aho-Corasick

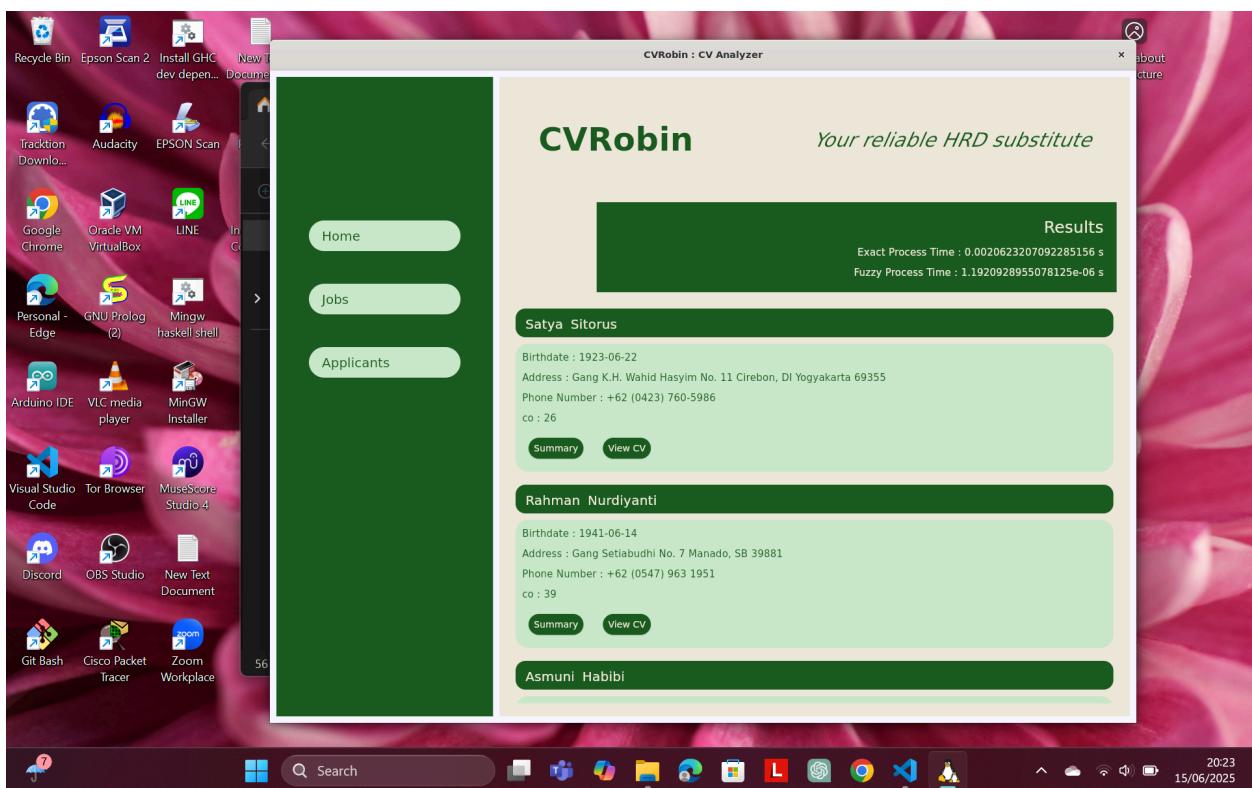
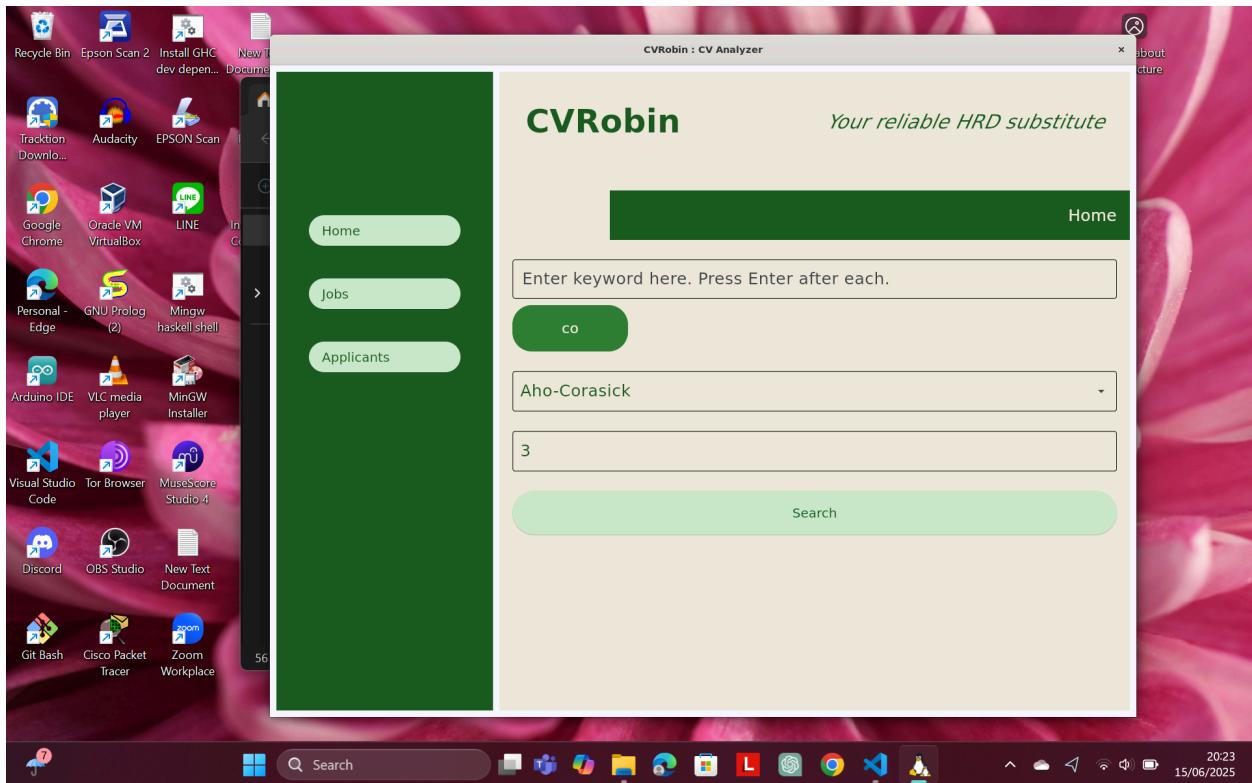
#### 1. Test case 1



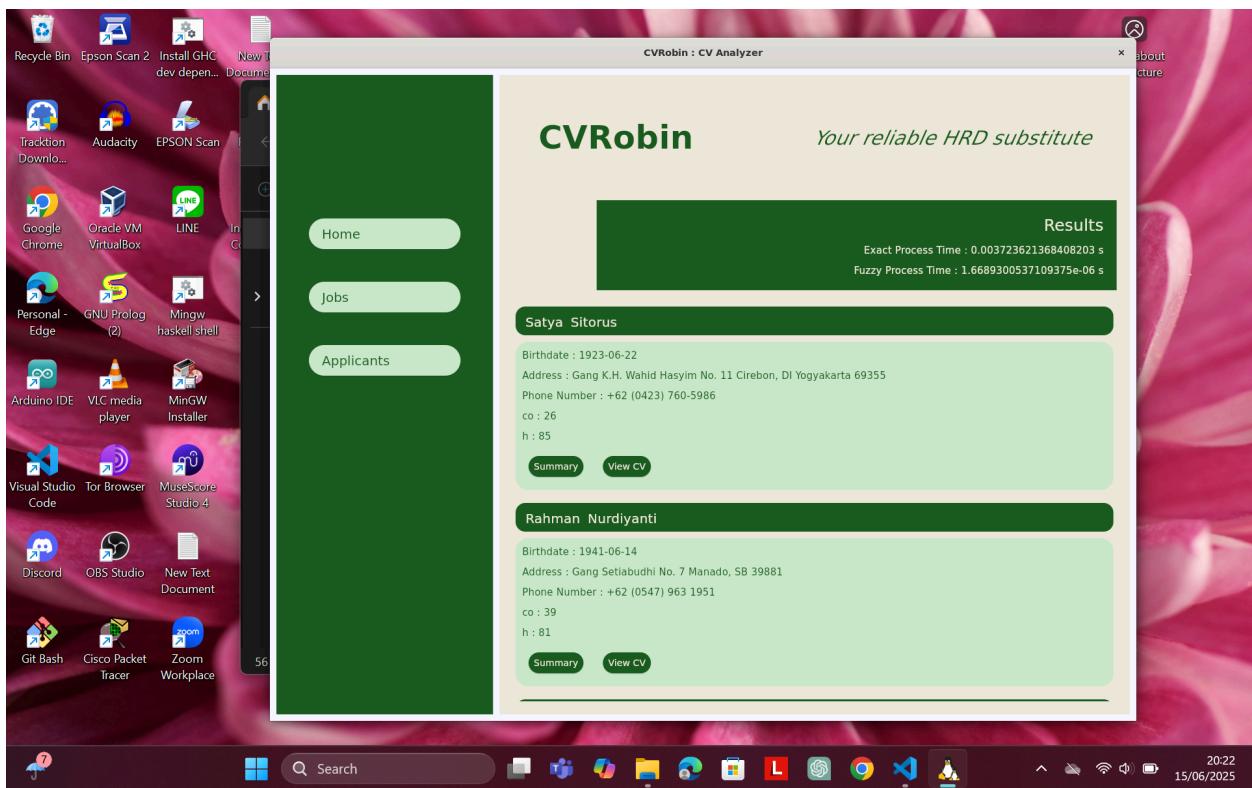
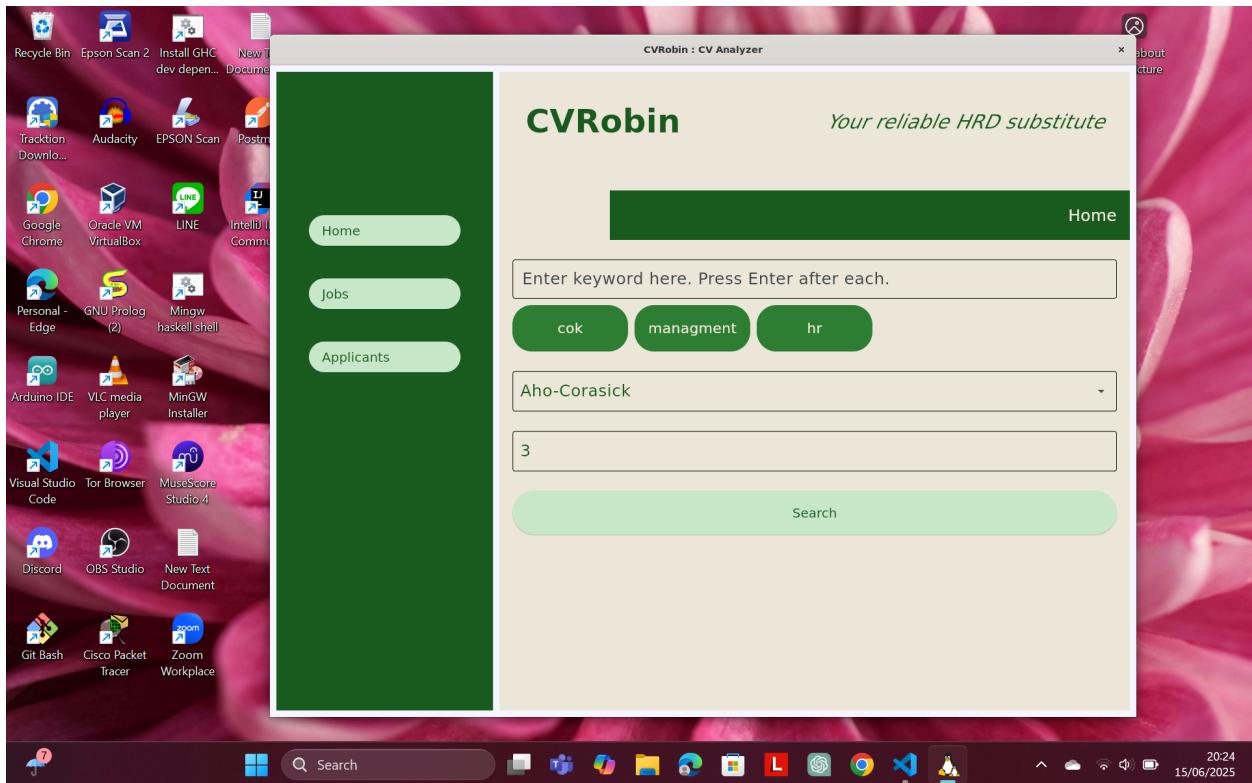
## 2. Test case 2



### 3. Test case 3



#### 4. Test case 4



#### 4.4. Analisis dan Pembahasan

Algoritma Aho-Corasick dirancang secara khusus untuk menjadi sangat efisien dalam mencari banyak kata kunci sekaligus. Sebelum pencarian dimulai, algoritma ini melalui tahap pra-pemrosesan dengan kompleksitas  $O(L)$ , di mana  $L$  adalah total panjang semua kata kunci. Algoritma membangun sebuah struktur data Tree yang dilengkapi dengan failure links, yang berfungsi sebagai mesin pencari. Setelah mesin ini terbentuk, proses pencarian menjadi sangat cepat dengan kompleksitas  $O(n)$ , di mana  $n$  adalah panjang teks, karena algoritma hanya perlu memeriksa seluruh teks satu kali tanpa pernah mundur. Hal ini membuat kompleksitas totalnya menjadi  $O(n + L)$ , menjadikannya pilihan terbaik untuk kasus penggunaan di mana sejumlah besar kata kunci perlu ditemukan dalam satu teks besar.

Algoritma Boyer-Moore (BM) dan Knuth-Morris-Pratt (KMP) pada dasarnya dioptimalkan untuk mencari satu kata kunci dalam sebuah teks. Masing-masing memiliki tahap pra-pemrosesan yang efisien dengan kompleksitas  $O(k)$ , di mana  $k$  adalah panjang kata kunci, untuk membuat tabel panduan (tabel bad character untuk BM dan tabel LPS untuk KMP). Proses pencarian untuk satu kata kunci juga bersifat linier, yaitu  $O(n)$ . Namun, ketika algoritma ini diadaptasi untuk mencari banyak kata kunci dengan cara melakukan iterasi, kompleksitas totalnya meningkat menjadi  $O(p \times (n + k))$ , dengan  $p$  sebagai jumlah kata kunci. Hal ini membuat dua algoritma ini menjadi kurang efisien dibandingkan Aho-Corasick ketika jumlah kata kunci yang dicari sangat banyak.

Algoritma Fuzzy Matching, yang diimplementasikan dengan Jarak Levenshtein, memiliki tujuan yang berbeda, yaitu menemukan kata yang "mirip" alih-alih yang identik persis. Akibatnya, algoritma ini memiliki kompleksitas yang paling berat. Tidak ada tahap pra-pemrosesan, dan proses pencarian melibatkan dua loop bersarang, satu untuk setiap kata dalam teks dan satu lagi untuk setiap kata kunci. Di dalam loop tersebut, algoritma menghitung jarak Levenshtein, yang memiliki kompleksitas  $O(w \times k)$ , di mana  $w$  dan  $k$  adalah panjang dari dua kata yang dibandingkan. Akibatnya, kompleksitas total algoritma ini menjadi sangat tinggi, yaitu  $O(W \times p \times w \times k)$ , dengan  $W$  sebagai jumlah kata dalam teks.

Untuk meningkatkan robustisitas dan menangani ketidaksesuaian data seperti kesalahan pengetikan, sistem ini dilengkapi dengan mekanisme pencocokan kabur

menggunakan Jarak Levenshtein. Algoritma ini mengukur jumlah perbedaan minimum antara dua kata dan efektif dalam mengidentifikasi kemiripan berdasarkan threshold yang telah ditentukan. Karena kompleksitasnya yang kuadratik  $O(m \times n)$ , penggunaannya dioptimalkan untuk perbandingan kata-kata individual guna menjaga efisiensi waktu.

Logika parsing CV dirancang secara berlapis sistem pertama-tama mencoba mengidentifikasi bagian-bagian dokumen yang eksplisit (seperti 'Education' atau 'Skill'), dan jika gagal, ia menerapkan strategi heuristik pada keseluruhan teks. Kombinasi ini memastikan sistem dapat beradaptasi dengan berbagai format CV yang tidak terstruktur.

Implementasi keempat algoritma yang digunakan dilakukan dalam bahasa Python dengan fokus pada optimasi performa. Algoritma Aho-Corasick menggunakan struktur data finite state automaton dengan failure function untuk pencarian multi-pattern yang efisien. Algoritma Boyer-Moore memanfaatkan bad character heuristic untuk melakukan lompatan karakter yang tidak cocok. Algoritma KMP menggunakan tabel border untuk menghindari perbandingan karakter yang tidak perlu. Sementara itu, implementasi fuzzy matching menggunakan dynamic programming untuk menghitung jarak edit Levenshtein dengan matriks dua dimensi.

Pemilihan algoritma yang tepat untuk setiap tugas Aho-Corasick untuk multi-pencarian dan Levenshtein untuk toleransi kesalahan menghasilkan sistem analisis yang kuat dan komprehensif. AC membangun sebuah program state tunggal dari semua kata kunci, memungkinkannya menemukan semua pola hanya dalam satu kali pemindaian teks, sebuah keunggulan performa yang signifikan dibandingkan menjalankan BM atau KMP secara berulang.

## BAB V

### KESIMPULAN DAN SARAN

#### 5.1. Kesimpulan

Dalam tugas besar ini, penulis telah mengembangkan sebuah sistem pencarian dan pencocokan pelamar kerja berbasis CV yang mampu mengekstraksi informasi dari dokumen PDF, menyimpan data secara terstruktur di MySQL, serta melakukan pencarian berdasarkan kata kunci dengan tingkat akurasi yang tinggi. Sistem ini dilengkapi dengan algoritma pencocokan string seperti Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick (AH) untuk exact matching, serta Levenshtein Distance untuk fuzzy matching, sehingga mampu mengakomodasi variasi penulisan dan kesalahan input pengguna. Selain itu, penggunaan *regular expression* memungkinkan sistem untuk mengekstrak informasi penting seperti keahlian, pengalaman kerja, dan riwayat pendidikan secara otomatis dari teks CV.

Secara keseluruhan, proyek ini memberikan pengalaman praktis dalam penerapan algoritma pencocokan string, ekstraksi teks, pengelolaan basis data, serta pengembangan antarmuka desktop. Sistem yang dikembangkan diharapkan dapat menjadi prototipe awal dalam pemrosesan otomatis data pelamar kerja, serta memberikan kontribusi terhadap efisiensi proses rekrutmen di era digital.

#### 5.2. Saran

Program yang kami buat masih jauh dari sempurna. Untuk kedepannya, bisa lebih memperhatikan struktur program, flow program yang lebih jelas, dan juga menyisihkan waktu lebih banyak untuk bug fixing secara total.

#### 5.3. Komentar

Kami mengucapkan terima kasih kepada asisten-asisten yang bertanggung jawab atas pelaksanaan tugas besar ini dan karena telah memberikan kesan yang terbaik untuk tugas pertama di Teknik Informatika ITB juga memberi gambaran akan bagaimana tugas-tugas besar lain yang akan dihadapi kedepannya. Kami juga mengucapkan terima kasih kepada mahasiswa Teknik Informatika ITB lain yang telah memberikan semangat dan dukungan sehingga pada akhirnya tugas besar ini dapat selesai dengan baik.

#### 5.4. Refleksi

Tugas besar ini telah memberikan kami pengalaman yang sangat mengesankan dari senang hingga tekanan. Kami menghadapi berbagai macam kendala seperti alokasi waktu yang kurang efektif akibat kesibukannya masing-masing, belum terbiasa dengan penerapan dari algoritma *pattern matching* itu sendiri, dan kinerja dari kami sendiri yang kurang baik. Untuk kedepannya, diharapkan tugas besar ini bisa menjadi motivasi untuk kami agar kami lebih berusaha kedepannya baik dalam penggerjaan tugas maupun menghadapi hidup

## LAMPIRAN

Link Repository GitHub:

[BrianHadianSTEI23/CVRobin: An implementation CV pattern identifier resembling ATS system using pattern-matching algorithm. Crafted by 3 Computer Science Undergraduate from Bandung Institute of Technology. Made with love](https://github.com/BrianHadianSTEI23/CVRobin)

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 Knuth-Morris-Pratt (KMP) dan Boyer-Moore (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 <i>applicant</i> .	✓	
10	Membuat bonus algoritma Aho-Corasick.	✓	
11	Membuat bonus video dan diunggah pada Youtube.	✓	

Link video: [https://youtu.be/D\\_QdfQQqOig?feature=shared](https://youtu.be/D_QdfQQqOig?feature=shared)

## DAFTAR PUSTAKA

[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)

[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)

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-(2025)-Bagian1.pdf)

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/26-Program-Dinamis-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/26-Program-Dinamis-(2025)-Bagian2.pdf)