

# **Laporan Tugas Besar 3 IF2211 - Strategi Algoritma**

## **Semester II Tahun Akademik 2024/2025**

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



*Disusun oleh:*

Refki Alfarizi	13523002
M. Jibril Ibrahim	13523085
Nayaka Ghana Subrata	13523090

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

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>DAFTAR GAMBAR.....</b>	<b>4</b>
<b>DAFTAR TABEL.....</b>	<b>5</b>
<b>BAB I</b>	
<b>DESKRIPSI TUGAS.....</b>	<b>6</b>
1.1 Deskripsi Tugas.....	6
<b>BAB II</b>	
<b>LANDASAN TEORI.....</b>	<b>8</b>
2.1 Knuth Morris Pratt.....	8
2.2 Boyer Moore.....	9
2.3 Aho Corasick.....	11
2.4 Regular Expression (regex).....	14
2.5 Jarak Levenshtein (Levenshtein Distance).....	15
2.6 Infrastruktur Aplikasi.....	16
2.6.1 Bahasa Pemrograman dan GUI.....	16
2.6.2 Database.....	16
2.6.3 Advanced Encryption Standard (AES).....	17
<b>BAB III</b>	
<b>ANALISIS PEMECAHAN MASALAH.....</b>	<b>19</b>
3.1 Langkah Pemecahan Masalah.....	19
<b>3.2 Proses Pemetaan Masalah Inti (Pattern Matching).....</b>	<b>20</b>
<b>3.3 Fitur Fungsional dan arsitektur aplikasi.....</b>	<b>22</b>
3.3.1 Fitur pemilihan algoritma pencocokkan string.....	22
3.3.2 Fitur pemilihan jumlah curriculum vitae yang ingin dicari.....	22
3.3.3 Fitur pencarian single query atau multiple query.....	22
3.3.4 Fitur menampilkan ringkasan dari applicant yang cocok.....	22
3.3.5 Fitur menampilkan cv dari applicant yang cocok.....	22
3.3.6 Arsitektur Aplikasi.....	22
<b>3.4 Ilustrasi Kasus.....</b>	<b>23</b>
<b>BAB IV</b>	
<b>IMPLEMENTASI DAN PENGUJIAN.....</b>	<b>26</b>
4.1 Spesifikasi Teknis Program.....	26
4.1.1 Struktur Data.....	26
4.1.2 Fungsi dan Prosedur.....	28
4.1.3 Algoritma.....	32
4.2 Tata Cara Penggunaan Program.....	39
4.3 Hasil Uji.....	43
4.3.1 Pencarian kata kunci “Chef” dengan algoritma KMP.....	43
4.3.2 Pencarian kata kunci “Chef” dengan algoritma Boyer Moore.....	43
4.3.3 Pencarian kata kunci “Chef” dengan algoritma Aho Corasick.....	44
4.3.4 Pencarian kata kunci “Ghana” dengan algoritma KMP.....	44
4.3.5 Pencarian kata kunci “Ghana” dengan algoritma Boyer Moore.....	45

4.3.6 Pencarian kata kunci “Ghana” dengan algoritma Aho Corasick.....	45
4.3.7 Pencarian kata kunci “Chef, cuisine, style” dengan algoritma KMP.....	46
4.3.8 Pencarian kata kunci “Chef, cuisine, style” dengan algoritma Boyer Moore.....	47
4.3.9 Pencarian kata kunci “Chef, cuisine, style” dengan algoritma Aho Corasick.....	47
4.3.10 Pencarian kata kunci “hahaha” dengan algoritma KMP.....	48
4.3.11 Pencarian kata kunci “hahaha” dengan algoritma Boyer Moore.....	48
4.3.12 Pencarian kata kunci “hahaha” dengan algoritma Aho Corasick.....	49
4.4 Analisis Hasil Uji.....	49
<b>BAB V</b>	
<b>PENUTUP.....</b>	<b>51</b>
5.1 Kesimpulan.....	51
5.2 Saran.....	51
5.3 Refleksi.....	52
<b>LAMPIRAN.....</b>	<b>53</b>
<b>DAFTAR PUSTAKA.....</b>	<b>54</b>
<b>AKHIR KATA.....</b>	<b>55</b>

## DAFTAR GAMBAR

Gambar 1.1 CV ATS dalam Dunia Kerja.....	6
Gambar 2.1 Contoh proses pencocokkan string pada algoritma Knuth-Morris-Pratt.....	8
Gambar 2.2 Contoh tabel fungsi pinggiran (border function).....	9
Gambar 2.3 Contoh proses pencocokkan string pada algoritma Boyer-Moore.....	10
Gambar 2.4 Contoh pencocokkan string pada algoritma Aho-Corasick.....	11
Gambar 2.5 Contoh pembentukan struktur data pada algoritma Aho-Corasick.....	12
Gambar 2.6 Proses pencocokkan regex.....	14
Gambar 2.7 Contoh beberapa notasi umum regex.....	15
Gambar 2.8 Implementasi Jarak Levenshtein pada permainan “Word Ladder”.....	15
Gambar 2.9 Skema enkripsi dengan menggunakan Algoritma Rijndael.....	18
Gambar 4.1 laman menu utama.....	41
Gambar 4.2 laman pencarian cv.....	41
Gambar 4.3 Hasil pencarian cv.....	42
Gambar 4.4 laman ringkasan cv.....	42

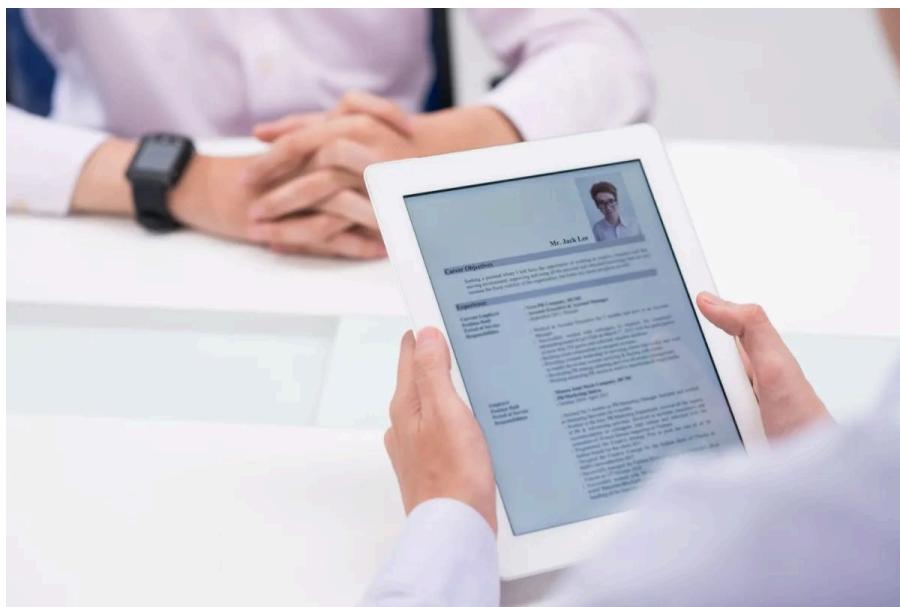
## **DAFTAR TABEL**

Tabel 1 Hasil ekstraksi dari pdf.....	24
Tabel 2 Poin yang dikerjakan.....	53

# BAB I

## DESKRIPSI TUGAS

### 1.1 Deskripsi Tugas



**Gambar 1.1 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.

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

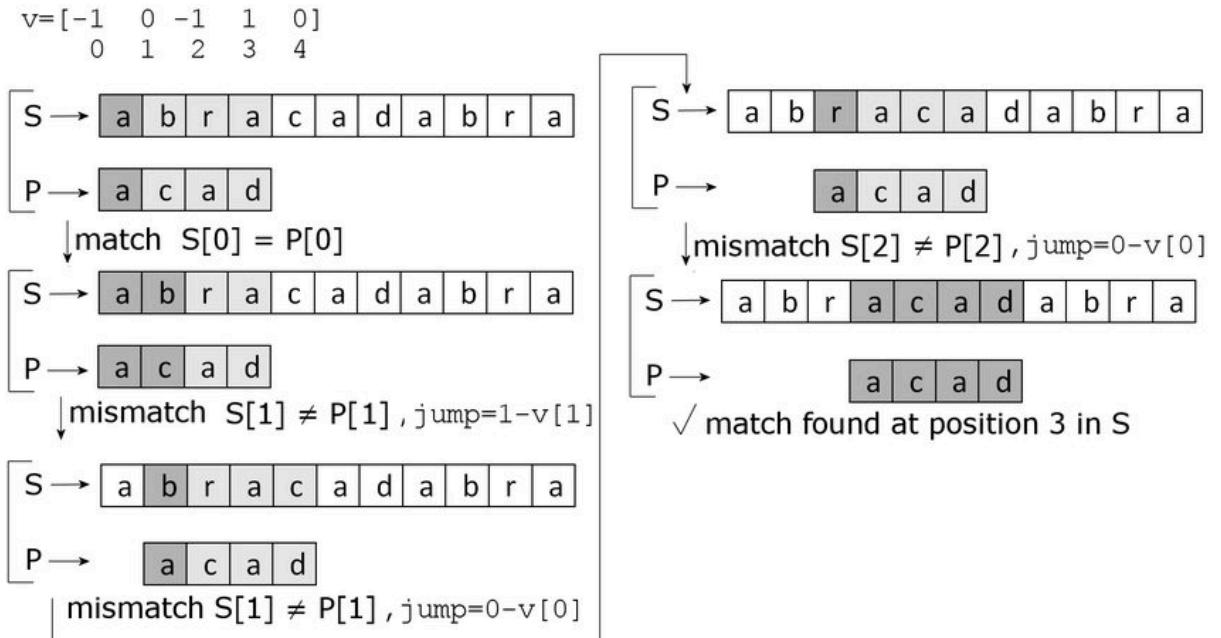
Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas kandidat melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali profil pelamar secara lengkap hanya dengan menggunakan CV digital.

## BAB II

### LANDASAN TEORI

#### 2.1 Knuth Morris Pratt

**Algoritma Knuth Morris Pratt (KMP)** termasuk ke dalam salah satu dari algoritma pencocokan *string* (*string matching*) yang ditemukan oleh James H. Morris, Donald Knuth (yang menemukannya secara independen seminggu kemudian dari teori automata), dan Vaughan Pratt yang melakukan *technical report* pada tahun 1970.



**Gambar 2.1** Contoh proses pencocokkan *string* pada algoritma Knuth-Morris-Pratt  
(Sumber:

<https://www.researchgate.net/publication/319954837/figure/fig4/AS:631660399845377@1527610983269/The-Knuth-Morris-Pratt-matching-process-for-the-example-in-Figure-21.png>

Algoritma Knuth-Morris-Pratt (KMP) menggeser pola pencarian untuk menghindari pemeriksaan karakter yang tidak perlu, tanpa mengurangi keakuratan pencocokan string. Kunci dari efisiensi ini terletak pada penentuan seberapa jauh pola harus digeser.

Pergeseran pola terbaik ditentukan berdasarkan informasi internal dari pola itu sendiri. Secara spesifik, pergeseran yang optimal adalah panjang dari prefix pola yang juga merupakan suffix dari bagian pola yang telah dibandingkan (yaitu,  $P[0\dots j-1]$  adalah prefix dari  $P$ , dan suffix dari  $P[1\dots j-1]$ ).

Untuk menghitung pergeseran ini, KMP menggunakan fungsi pinggiran (*border function*). Fungsi pinggiran memberikan panjang prefix terpanjang dari pola ( $P[0\dots k]$ ) yang juga menjadi *suffix* dari bagian pola yang sama ( $P[1\dots k]$ ), di mana  $k = j-1$ . Dengan memanfaatkan fungsi pinggiran, algoritma KMP dapat menentukan jumlah pergeseran pola yang tepat untuk mencegah pencocokan yang tidak efisien atau "membuang-buang" waktu, sehingga proses pencarian string menjadi jauh lebih cepat dan efektif.

#### Constructing prefix table for Pattern **ABCDABD**

j	0	1	2	3	4	5	6
substring 0 to j	A	AB	ABC	ABCD	ABCDA	ABCDAB	ABCDABD
longest prefix-suffix match	none	none	none	none	A	AB	none
Length of prefix-suffix (Number of characters)	0	0	0	0	1	2	0

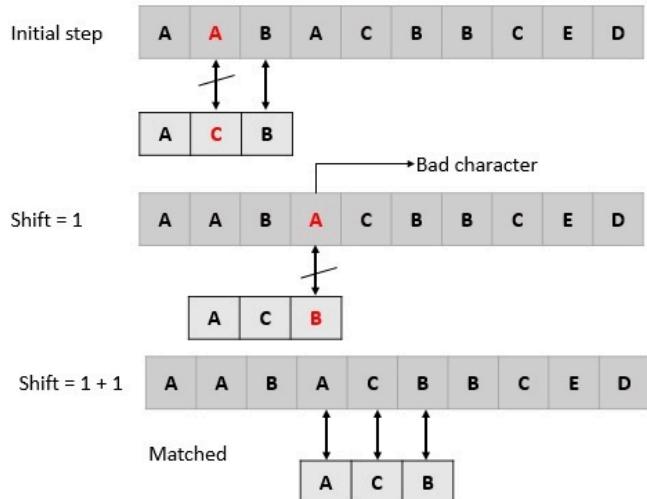
**Gambar 2.2** Contoh tabel fungsi pinggiran (*border function*)

(Sumber: <https://i.sstatic.net/XutaG.png>)

Kompleksitas Algoritma Knuth-Morris-Pratt memiliki karakteristik yang lebih konsisten dibandingkan algoritma lainnya. Algoritma KMP memiliki kompleksitas waktu  $O(n+m)$  baik untuk *worst-case* maupun *average-case*, di mana  $n$  adalah panjang teks dan  $m$  adalah panjang pola. Kompleksitas linear ini dicapai melalui *preprocessing* yang membangun *failure function* (atau *prefix function*) dalam waktu  $O(m)$ , kemudian diikuti dengan fase pencarian yang membutuhkan waktu  $O(n)$ . Keunggulan utama KMP adalah konsistensinya, yakni algoritma ini tidak pernah mundur dalam teks dan setiap karakter teks hanya diperiksa maksimal dua kali. Kompleksitas ruang yang dibutuhkan adalah  $O(m)$  untuk menyimpan *failure function*, yang membuat KMP sangat *predictable* dan cocok untuk aplikasi *real-time*.

## 2.2 Boyer Moore

**Algoritma Boyer-Moore** merupakan metode *pattern-matching* yang dikembangkan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977. Berbeda dengan algoritma pencarian *string* sebelumnya, Boyer-Moore memulai pencocokan karakter dari ujung kanan pola (*pattern*) sehingga dapat memanfaatkan informasi kegagalan yang terjadi paling akhir untuk menggeser pola lebih jauh dan mengurangi jumlah perbandingan secara signifikan.



**Gambar 2.3** Contoh proses pencocokkan *string* pada algoritma Boyer-Moore

(Sumber: [https://www.tutorialspoint.com/data\\_structures\\_algorithms/images/pattern\\_boyer.jpg](https://www.tutorialspoint.com/data_structures_algorithms/images/pattern_boyer.jpg))

Notasi dari algoritma Boyer-Moore dapat didefinisikan sebagai berikut:

Teks:  $T = T[0..n - 1]$ , Panjang  $n$

Pola:  $P = P[0..m - 1]$ , Panjang  $m$

dengan fungsi *last occurrence*:

$$\ell(c) = \begin{cases} \max\{i \mid 0 \leq i < m, P[i] = c\}, & c \in P, \\ -1, & c \notin P. \end{cases}$$

Selain itu, algoritma Boyer-Moore dilakukan dengan *Looking-Glass Scanning*, yakni dimulai dengan menggeser pola ke posisi  $s=0$ . Untuk setiap  $s \leq n-m$ , algoritma akan melakukan perbandingan dengan membandingkan  $P[m-1] = T[s+m-1]$ , kemudian  $P[m-2] = T[s+m-2]$ , dan seterusnya hingga salah satu dari dua kondisi terpenuhi:

- Sementara, yakni semua karakter cocok maka pola ditemukan di posisi  $s$
- Terjadi mismatch di indeks pola  $j$ , yang mengharuskan algoritma untuk melanjutkan pencarian ke posisi berikutnya.

Jika *mismatch* terjadi pada  $j$ , artinya  $P[j] \neq T[s+j]$  dan karakter teksnya  $x = T[s+j]$ , maka dilakukan perhitungan sebagai berikut:

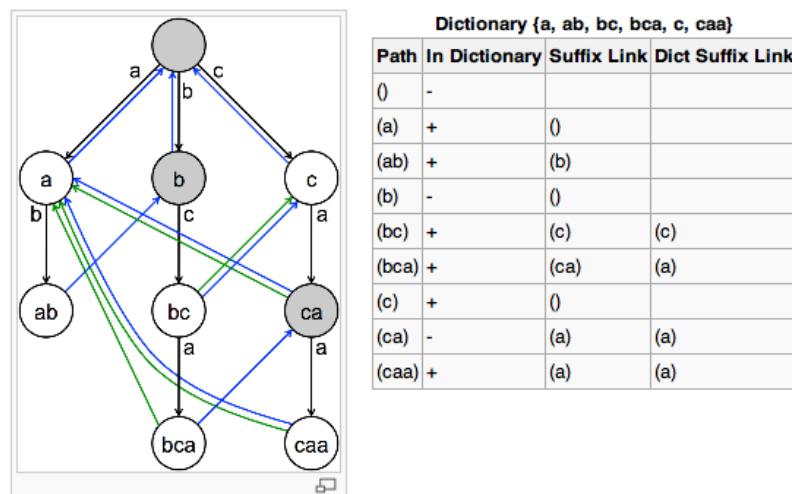
$$k = \ell(x), \quad \Delta = j - k, \quad s += \max(1, \Delta).$$

Terdapat tiga kasus yang dapat terjadi. **Case 1** ( $k \leq j$ ): geser sejauh  $j-k > 0$  agar kemunculan terakhir  $x$  di  $P$  tepat di bawah posisi  $s+j$ . **Case 2** ( $k > j$ ):  $j-k \leq 0$ , sehingga geser minimal 1. **Case 3** ( $k = -1$ ): geser  $j-(-1) = j+1$ , melewati seluruh jendela gagal.

Kompleksitas algoritma Boyer-Moore memiliki karakteristik yang berbeda tergantung pada kondisi input dan heuristik yang digunakan. Dalam skenario *worst-case* tanpa menggunakan heuristik *good-suffix*, algoritma ini memiliki kompleksitas waktu  $O(mn)$ , di mana  $m$  adalah panjang pola dan  $n$  adalah panjang teks. Kondisi terburuk ini dapat terjadi ketika algoritma harus melakukan perbandingan yang ekstensif pada setiap posisi tanpa mendapat keuntungan dari kemampuan melompat yang optimal. Namun, pada kondisi *average-case* dengan teks yang bersifat acak, algoritma Boyer-Moore menunjukkan performa yang jauh lebih baik dengan kompleksitas  $\Theta(n/m)$ . Kompleksitas *average-case* yang menguntungkan ini dicapai karena algoritma dapat memanfaatkan informasi dari karakter yang tidak cocok untuk melakukan lompatan yang signifikan, sehingga tidak perlu memeriksa setiap karakter dalam teks. Dalam praktiknya, performa algoritma biasanya mendekati linear terhadap panjang teks, yang membuatnya sangat efisien untuk pencarian *string* pada teks berukuran besar dengan pola yang relatif panjang.

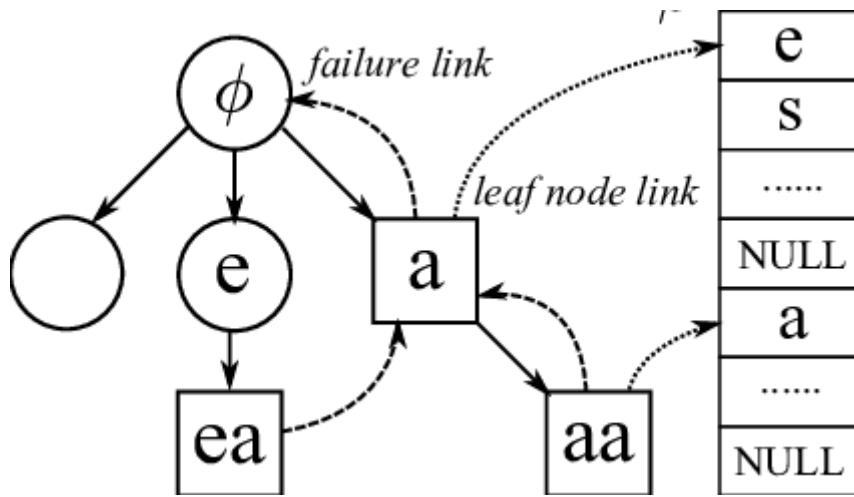
### 2.3 Aho Corasick

**Algoritma Aho-Corasick** merupakan algoritma pencarian *string* revolusioner yang dikembangkan oleh Alfred V. Aho dan Margaret J. Corasick pada tahun 1975. Algoritma ini dirancang khusus untuk mengatasi tantangan pencarian *multiple pattern*, yaitu mencari beberapa pola *string* secara bersamaan dalam satu teks input. Berbeda dengan algoritma pencarian *string* tradisional yang hanya dapat mencari satu pola pada satu waktu, Aho-Corasick mampu melakukan pencarian terhadap seluruh kamus (*dictionary*) yang berisi kumpulan *finite string target* dalam sekali proses scanning.



**Gambar 2.4** Contoh pencocokan *string* pada algoritma Aho-Corasick  
(Sumber: <https://i.sstatic.net/52b7L.png>)

Struktur Data Algoritma Aho-Corasick dibangun berdasarkan kombinasi dua komponen fundamental: **Trie** dan **Failure Function**. Struktur ini memungkinkan algoritma untuk mencari *multiple pattern* secara efisien dalam satu kali *scanning*.



**Gambar 2.5** Contoh pembentukan struktur data pada algoritma Aho-Corasick  
(Sumber:

<https://www.researchgate.net/publication/220873369/figure/fig1/AS:652199600328717@1532507910921/Rule-Index-based-on-Aho-Corasick-Tree.png>

**Trie (Prefix tree)** merupakan fondasi utama dari algoritma Aho-Corasick. Struktur ini menyimpan semua *pattern* dari *dictionary* dengan cara yang memungkinkan berbagi *prefix* yang sama antar *pattern*. Setiap *node* dalam trie merepresentasikan satu karakter, dan *path* dari *root* ke suatu *node* membentuk *prefix* dari satu atau lebih *pattern*. *Node-node* terminal ditandai sebagai "output node" yang menunjukkan bahwa suatu *pattern* telah ditemukan. Konstruksi trie dilakukan dengan memasukkan semua *pattern* satu per satu, di mana setiap karakter dari *pattern* membentuk *edge* atau transisi dalam pohon.

**Failure function (failure link)** merupakan kunci yang membedakan Aho-Corasick dari trie biasa. Setiap *node* dalam trie memiliki *failure link* yang menunjuk ke *node* lain dalam trie, yang merepresentasikan *suffix* terpanjang dari *string* yang berakhir di *node* tersebut yang juga merupakan *prefix* dari *pattern* lain dalam *dictionary*. Ketika terjadi mismatch pada suatu *node*, algoritma tidak perlu kembali ke *root*, melainkan dapat mengikuti *failure link* untuk melanjutkan pencarian tanpa kehilangan informasi yang telah diproses. *Failure function* dihitung menggunakan pendekatan *breadth-first search*, di mana *failure link* untuk setiap *node* dihitung berdasarkan *failure link* dari *node parent*-nya.

**Output function** melengkapi struktur data dengan menyimpan informasi tentang semua *pattern* yang berakhir pada setiap *node*. Tidak hanya *pattern* yang berakhir langsung di *node* tersebut, tetapi juga *pattern* yang berakhir di *node-node* yang dapat dicapai melalui *failure link*. Hal ini memungkinkan deteksi *overlapping matches*, di mana *multiple pattern* dapat ditemukan pada posisi yang sama dalam teks. Kombinasi ketiga komponen ini

menciptakan automaton yang dapat memproses teks input dalam waktu linear sambil mendeteksi semua kemunculan *pattern* dari *dictionary* secara bersamaan.

Berikut ini adalah representasi matematis dari setiap struktur data.

*failure function:*

$$f(q) = \max\{k \mid k < |q| \text{ dan } q[0..k-1] \text{ adalah suffix dari } q[0..|q|-1]\}$$

*goto function:*

$$g(q, a) = \begin{cases} \text{next state,} & \text{jika ada transisi dari state } q \text{ dengan karakter } a \\ \text{fail,} & \text{jika tidak ada transisi} \end{cases}$$

*output function:*

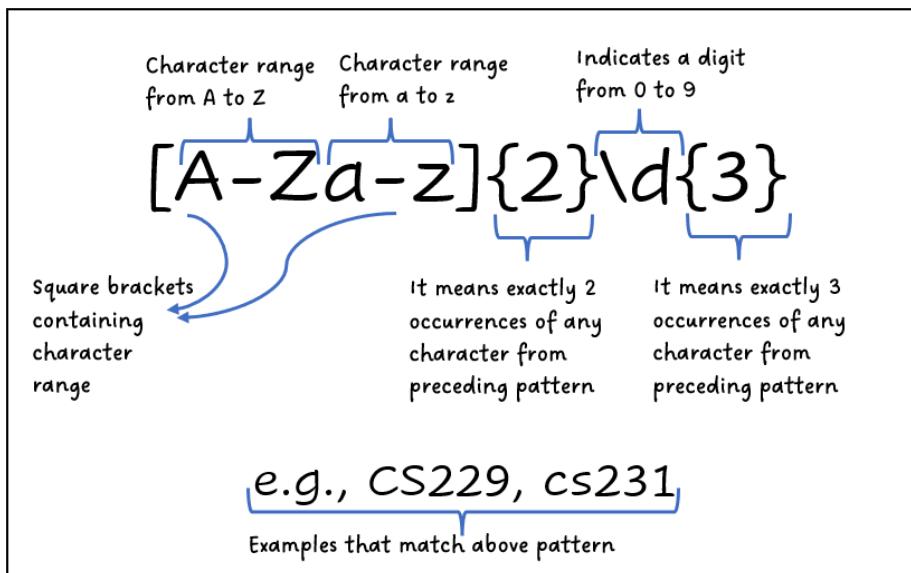
$$\text{output}(q) = \{P_i \mid P_i \text{ berakhir di state } q\} \cup \bigcup_{s \in \text{fail-path}(q)} \text{output}(s)$$

*automaton transition function:*

$$\delta(q, a) = \begin{cases} g(q, a), & \text{jika } g(q, a) \neq \text{fail} \\ \delta(f(q), a), & \text{jika } g(q, a) = \text{fail} \end{cases}$$

Kompleksitas Algoritma Aho-Corasick terdiri dari dua fase. Fase *preprocessing* membutuhkan waktu  $O(m)$  untuk membangun trie, *failure function*, dan *output function*, di mana  $m$  adalah total panjang semua *pattern* dalam *dictionary*. Fase *searching* memiliki kompleksitas  $O(n+z)$ , dengan  $n$  sebagai panjang teks input dan  $z$  sebagai jumlah *match* yang ditemukan. Kompleksitas ruang adalah  $O(m)$  untuk menyimpan struktur data automaton. Secara keseluruhan, algoritma Aho-Corasick mencapai kompleksitas total  $O(m+n+z)$ , yang merupakan kompleksitas linear terhadap ukuran input dan output, menjadikannya sangat efisien untuk pencarian *multiple pattern* karena setiap karakter teks hanya diproses sekali tanpa *backtracking*.

## 2.4 Regular Expression (regex)



Gambar 2.6 Proses pencocokan regex

(Sumber:[https://miro.medium.com/v2/resize:fit:823/1\\*u4NbHgv-TzEE5G6xXnHCbQ.png](https://miro.medium.com/v2/resize:fit:823/1*u4NbHgv-TzEE5G6xXnHCbQ.png))

**Regex**, singkatan dari **Regular Expression**, adalah sebuah pola yang digunakan untuk mencocokkan kombinasi karakter dalam sebuah *string*. Bisa dibilang, regex adalah implementasi dan representasi dari bahasa formal, yang memungkinkan kita untuk mendefinisikan pola teks yang kompleks.

Dengan pola Regexp, kita bisa membuat berbagai kombinasi kata atau bahkan kalimat sesuai dengan definisi pola yang diinginkan. Pola ini kemudian berfungsi sebagai "filter" untuk **menentukan apakah suatu kata atau kalimat sesuai dengan pola regex yang telah didefinisikan**. Ini sangat berguna untuk berbagai tugas seperti validasi input, pencarian dan pengantian teks, hingga *parsing* data.

Untuk membangun pola regex, digunakan beberapa notasi atau karakter khusus. Notasi-notasi inilah yang memberikan kekuatan dan fleksibilitas pada regex untuk mendefinisikan pola pencarian yang sangat spesifik. Beberapa karakter/notasi umum yang digunakan dalam pola regex meliputi:

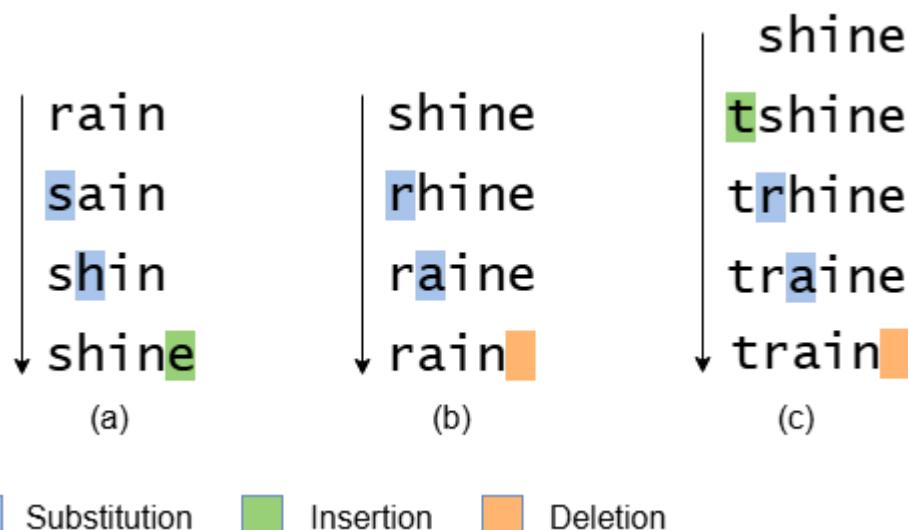
Metacharacter	Description
<code>?</code>	Matches <b>zero or one</b> time
<code>*</code>	Matches <b>zero or more</b> times
<code>+</code>	Matches <b>one or more</b> times. In other words, it matches at least one time
<code>{n}</code>	Matches <b>exactly n</b> times
<code>{n, m}</code>	Matches <b>at least n and at most m</b> times
<code>{n,}</code>	Matches <b>at least n times (no upper bound)</b>
<code>,m}</code>	Matches <b>at most m times (lower bound is zero)</b>

**Gambar 2.7** Contoh beberapa notasi umum regex  
 (Sumber: [https://miro.medium.com/v2/resize:fit:1400/1\\*Cbye2MECJ8APu\\_r8uCiFOw.png](https://miro.medium.com/v2/resize:fit:1400/1*Cbye2MECJ8APu_r8uCiFOw.png))

## 2.5 Jarak Levenshtein (Levenshtein Distance)

Algoritma pencocokan *string* seperti KMP (Knuth-Morris-Pratt) dan BM (Boyer-Moore) serta Aho-Corasick dirancang untuk pencocokan eksak. Artinya, algoritma ini hanya dapat menentukan apakah sebuah pola tertentu benar-benar ada dalam suatu teks secara keseluruhan. Jika ada sedikit saja perbedaan (*typo*), pencocokan akan gagal.

Namun, dalam banyak kasus, pencocokan eksak tidak selalu memungkinkan atau diinginkan. Terkadang, perlu mencari pola yang mirip, meskipun tidak identik. Di sinilah algoritma pencocokan *string* non-eksak berperan penting. Salah satu algoritma yang paling populer adalah **Levenshtein Distance**.



**Gambar 2.8** Implementasi Jarak Levenshtein pada permainan “Word Ladder”  
 (Sumber: <https://devopedia.org/images/article/213/5510.1567535069.svg>)

**Jarak Levenshtein**, yang dinamai dari Vladimir Levenshtein (1965), adalah metrik untuk mengukur perbedaan antara dua *string* berdasarkan operasi *insertion*, *deletion*, dan *substitution*. Dalam konteks *fuzzy-matching*, algoritma ini digunakan untuk menemukan *substring* yang "hampir cocok".

Algoritma ini membentuk matriks  $d[i,j]$  berdimensi  $(n+1) \times (m+1)$ , di mana  $n$  adalah panjang *string* pertama dan  $m$  adalah panjang *string* kedua.

$$d[i, j] = \min \begin{cases} d[i - 1, j] + 1, \\ d[i, j - 1] + 1, \\ d[i - 1, j - 1] + 1_{S_{i-1} \neq P_{j-1}} \end{cases}$$

Dengan kondisi batas:  $d[0,j]=j$  dan  $d[i,0]=i$ .

Untuk setiap posisi  $s$  di teks ( $0 \leq s \leq n-m$ ), algoritma mengambil substring  $S[s..s+m-1]$  dan menghitung  $d[m,m]$ . Jika  $d[m,m] \leq \tau$  (threshold), hasilnya dicatat sebagai  $(s, d[m,m])$ . Selama pengisian baris ke- $i$ , jika  $\min_j d[i,j] > \tau$ , perhitungan dapat dihentikan lebih awal (*pruning*) karena jarak tidak mungkin turun di bawah  $\tau$ .

Kompleksitas algoritma *Levenshtein Distance* adalah  $O(m^2)$  per *window* (atau per pasangan *string*) tanpa optimisasi, di mana  $m$  adalah panjang pola. Ini berarti waktu komputasi bertambah seiring kuadrat panjang pola. Namun, dengan optimisasi *pruning*, kinerja rata-rata algoritma ini menjadi jauh lebih cepat, terutama untuk ambang batas ( $\tau$ ) kemiripan yang kecil, karena banyak perhitungan yang tidak perlu bisa dihentikan lebih awal.

## 2.6 Infrastruktur Aplikasi

### 2.6.1 Bahasa Pemrograman dan GUI

Pada implementasinya, aplikasi menggunakan bahasa pemrograman Python sebagai bahasa utama dalam membuat dan mengembangkan aplikasi. Oleh karena itu, untuk menampilkan *Graphical User Interface* (GUI), digunakan PyQt karena mendukung implementasi GUI pada *environment* Python. Kelebihan dari penggunaan PyQt dibandingkan dengan penyedia GUI Python yang lain adalah fitur-fiturnya yang lebih lengkap, skalabilitasnya yang tinggi, dan ada fitur QT designer yang membuat GUI bisa didesain terlebih dahulu sebelum kode dibuat ke program.

Selain menggunakan PyQt sebagai penunjang GUI, aplikasi juga menggunakan *library* PyMuPDF yang akan melakukan ekstraksi *curriculum vitae* (cv) dari pdf ke penyimpanan memori yang dilakukan ketika *runtime*, sehingga akses data cv menjadi lebih efisien dan cepat karena tersimpan pada memori, dibandingkan dengan menyimpannya pada *disk*. Selain itu, kelebihan dari PyMuPDF adalah fiturnya yang cukup *straightforward*, membuat *library* tersebut lebih terasa ringan dibandingkan dengan *library* ekstraktor pdf Python yang lain.

### 2.6.2 Database

Selain melakukan pencocokan *string*, tentunya data-data terkait *curriculum vitae* (cv) sebaiknya disimpan pada suatu sistem penyimpanan data, salah satunya adalah dengan menyimpan data tersebut pada *Database Management System* (DBMS). Pada aplikasi ini, digunakan DBMS berupa MySQL yang disinkronisasi menggunakan MariaDB.

*Deployment* dari *database* dilakukan dengan menggunakan *docker image* dari MySQL, yang selanjutnya kredensial dan informasi terkait *database* akan disimpan pada file

.env. Untuk melakukan sinkronisasi antara data .env dan juga aplikasi terhadap DBMS, digunakan suatu *library* Python yang mendukung koneksi ke MySQL, yakni dengan menggunakan *library* mysql.connector.

Selain koneksi, aplikasi juga menyediakan fitur *seeding* baik secara mandiri maupun *load* dari suatu file .sql. Untuk *seeding* secara mandiri, digunakan *library* Python yakni Faker untuk melakukan *seeding* data dari *database* dengan *database* palsu yang disediakan oleh *library* Faker. Selain *seeding* mandiri, ada juga opsi untuk melakukan *load* data dari file .sql, yang mendukung pengisian *database* dengan menulis data secara *hardcode* pada file dengan ekstensi .sql, sehingga data bisa diatur sesuai dengan keinginan sebelum masuk ke tahap *production*.

### 2.6.3 Advanced Encryption Standard (AES)

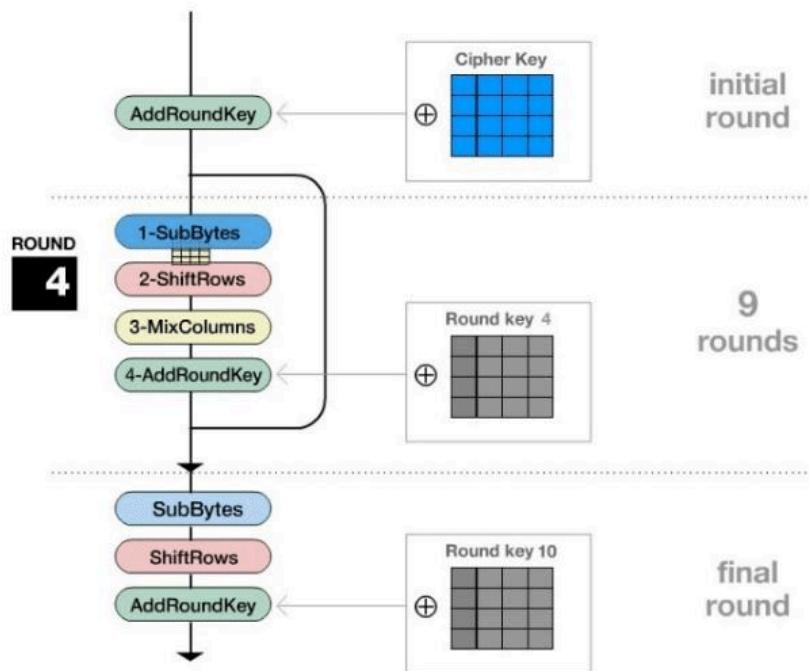
Untuk menjaga keamanan dan integritas data yang tersimpan pada *database*, aplikasi ini menyediakan skema enkripsi dengan menggunakan algoritma *Advanced Encryption Standard* (AES). AES merupakan algoritma enkripsi simetris yang menggunakan *private key* untuk melakukan enkripsi dan dekripsinya.

Dalam implementasinya, digunakan algoritma Rijndael (*Rijndael algorithm*). Proses ini dimulai dengan melakukan “key expansion” atau “key schedule”, di mana dari kunci 128-bit yang diberikan, sebelas “kunci putaran” (*round keys*) 128-bit terpisah diturunkan. Setiap kunci putaran ini akan digunakan dalam setiap langkah “AddRoundKey”.

Langkah selanjutnya adalah “Initial Key Addition”. Di sini, *bytes* dari kunci putaran pertama di-XOR dengan *bytes* dari “state” (representasi data yang sedang dienkripsi). Setelah itu, algoritma memasuki fase “Round” yang diulang sebanyak sepuluh kali: sembilan putaran utama ditambah satu “final”.

Dalam setiap putaran (kecuali “MixColumns” pada putaran terakhir), empat transformasi utama terjadi:

- SubBytes: Setiap *byte* dalam *state* disubstitusikan dengan *byte* lain menggunakan tabel pencarian yang disebut “S-box”.
- ShiftRows: Tiga baris terakhir dari matriks *state* digeser (*transposed*) beberapa kolom. Baris kedua digeser satu kolom, baris ketiga dua kolom, dan baris keempat tiga kolom. Ini memastikan bahwa *bytes* dari kolom yang berbeda saling berinteraksi.
- MixColumns: Matriks perkalian dilakukan pada kolom-kolom *state*, menggabungkan empat *bytes* di setiap kolom. Langkah ini meningkatkan difusi data, tetapi dilewati pada putaran final untuk alasan efisiensi dan keamanan.
- AddRoundKey: Terakhir, *bytes* dari kunci putaran saat ini di-XOR dengan *bytes* dari *state*. Ini mengintegrasikan kunci ke dalam proses enkripsi pada setiap putaran.



**Gambar 2.9** Skema enkripsi dengan menggunakan Algoritma Rijndael  
 (Sumber:

[https://www.researchgate.net/publication/313899040/figure/fig1/AS:464907083751424@1487853894\\_054/Rijndael-Encryption-Algorithm-2-2-Serpent-Encryption-algorithm-Serpent-Encryption.png](https://www.researchgate.net/publication/313899040/figure/fig1/AS:464907083751424@1487853894_054/Rijndael-Encryption-Algorithm-2-2-Serpent-Encryption-algorithm-Serpent-Encryption.png)

Untuk melakukan dekripsi, cukup dengan melakukan *inverse* pada masing-masing proses, tetapi karena ada penambahan *key* pada suatu proses, maka untuk melakukan dekripsi, tetap dibutuhkan kunci yang digunakan pada saat melakukan enkripsi pada data.

## BAB III

### ANALISIS PEMECAHAN MASALAH

#### 3.1 Langkah Pemecahan Masalah

Dalam sistem ATS yang akan dibangun, **permasalahan intinya adalah melakukan pencocokan pola** (*pattern matching*) pada teks CV yang disimpan pengguna, mulai dari menemukan kata kunci persis hingga menangani variasi ejaan kecil, serta mengekstrak informasi penting dari teks mentah CV secara otomatis. Secara ringkas, *problem statement* yang harus diselesaikan meliputi:

1. Menyimpan informasi *applicants* pada suatu model basis data dengan aman.
2. Ekstraksi teks mentah dari file PDF ke dalam format string yang siap diproses.
3. Pencarian kata kunci persis (*exact match*) untuk setiap *keyword* masukkan pengguna, sekaligus mendukung banyak *keyword* dalam satu pencarian.
4. Penanganan kegagalan *exact match* dengan melakukan pencarian *fuzzy-matching* agar typo kecil atau variasi ejaan tetap terdeteksi.
5. Ekstraksi informasi terstruktur (ringkasan, keahlian, pengalaman kerja, riwayat pendidikan) dari teks CV menggunakan pola Regular Expression.
6. Pengurutan dan penyajian hasil berdasarkan jumlah kecocokan dan relevansi, agar pengguna dapat segera melihat CV teratas.
7. Integrasi antarmuka sehingga pengguna dapat memilih algoritma *pattern-matching* dan meninjau ringkasan data secara interaktif.

Berikut uraian bagaimana setiap elemen *problem statement* dipecahkan:

1. Penyimpanan informasi *applicants* dengan aman

Dalam menyimpan data para *applicants*, kami menggunakan MySQL sebagai sistem manajemen basis data yang di-*containerized* pada Docker. Selain itu, untuk menjaga keamanan dan integritas data pribadi para *applicants*, aplikasi ini mengimplementasikan skema enkripsi dengan menggunakan algoritma *Advanced Encryption Standard* (AES). AES merupakan algoritma enkripsi simetris yang menggunakan *private key* untuk melakukan enkripsi dan dekripsinya.

2. Ekstraksi teks mentah dari PDF

Untuk melakukan ekstraksi teks dari CV berbentuk PDF, program memakai PyMuPDF untuk membuka dan mengekstrak setiap baris dan halaman pada PDF. Untuk memudahkan proses pengambilan informasi penting dengan RegEx dan pencocokan kata kunci, setiap hasil teks diolah menjadi dua tipe yaitu hasil RegEx dan hasil *pattern-matching*. Pemisahan ini dilakukan dengan alasan pengambilan

informasi dengan RegEx memerlukan struktur yang cukup jelas jika dibandingkan dengan teks untuk *pattern-matching* yang tidak *case sensitive* dan hanya dengan karakter ASCII.

3. Pencarian kata kunci persis (*Exact Match*)

Di lapisan pencarian, sistem mendukung tiga algoritma utama yaitu Knuth-Morris-Pratt, Boyer-Moore, dan Aho-Corasick. Pada sub bab berikutnya akan dijelaskan lebih rinci mengenai pemilihan algoritma tersebut sebagai solusi. Dengan demikian, pengguna dapat memilih metode paling sesuai berdasarkan pola dan karakteristik teks.

4. Penanganan kegagalan exact match dengan *fuzzy matching*

Jika suatu *keyword* tidak ditemukan secara persis, sistem secara otomatis melakukan *approximate matching* menggunakan Levenshtein Distance. Hanya *window*/subset teks dengan jarak edit di bawah ambang batas (threshold) yang diakui, sehingga typo kecil atau variasi ejaan tetap terdeteksi.

5. Ekstraksi informasi terstruktur via Regular Expression

Setelah proses pencocokan, bagian teks yang berisi ringkasan, daftar keahlian, pengalaman kerja, dan riwayat pendidikan diekstrak menggunakan pola Regular Expression. Setiap pola dirancang untuk menangkap format umum CV, misalnya *heading* “Skills” diikuti daftar kata kunci.

6. Pengurutan dan penyajian hasil

Untuk menampilkan kandidat paling relevan, sistem menghitung skor berdasarkan jumlah *exact match* lalu *fuzzy match* (prioritas *exact match*). Hasil kemudian diurutkan menurun sehingga CV dengan kecocokan terbanyak muncul di bagian atas daftar dengan batas jumlah hasil yang ditunjukkan sesuai masukkan dari pengguna.

7. Integrasi antarmuka interaktif

Semua langkah di atas dipanggil dari tampilan desktop (PyQt) yang dapat dijalankan melalui platform Windows maupun Linux. Pengguna dapat memasukkan keyword, memilih algoritma *exact match*, lalu melihat ringkasan hasil pencarian, informasi *applicant*, dan isi dari CV.

### 3.2 Proses Pemetaan Masalah Inti (*Pattern Matching*)

Sebagai titik awal, pencarian pola di dalam teks CV dapat dilakukan dengan metode *brute-force*, yakni memeriksa setiap posisi teks terhadap setiap karakter pola, satu per satu. Pendekatan ini memiliki kompleksitas waktu  $O(nm)$ , di mana  $n$  adalah panjang teks dan  $m$  panjang pola, karena pada setiap posisi teks upaya perbandingan dapat mencapai  $m$  kali. Meskipun implementasinya sederhana, pada dokumen CV yang cukup panjang dan banyak atau daftar keyword yang banyak, kinerja *brute-force* menjadi tidak praktis jika dibandingkan dengan algoritma yang lebih “cerdas”.

Knuth-Morris-Pratt (KMP) memanfaatkan konsep *failure function* atau *border function* untuk menghindari mundurnya pointer teks ketika terjadi mismatch. KMP memproses pola dalam waktu linear  $O(n+m)$  pada semua kasus, termasuk kasus terburuk, dengan membangun terlebih dahulu larak LPS (*longest proper prefix which is also suffix*). Keunggulan ini menjamin kestabilan performa bahkan ketika teks dan pola memiliki banyak pengulangan. Namun, KMP kurang optimal saat alfabet menjadi sangat besar, karena semakin banyak kemungkinan mismatch awal yang membuat lompatan yang dihasilkan cenderung kecil dan *overhead preprocessing* menjadi relatif mahal. Oleh karena itu, KMP dapat dipilih sebagai opsi ketika jaminan kompleksitas linear dan pola tunggal dengan panjang sedang diutamakan.

Boyer-Moore menawarkan heuristik yang berbeda, yaitu dengan memulai perbandingan dari ujung kanan pola sehingga dapat segera memanfaatkan informasi *mismatch* paling akhir untuk melakukan loncatan besar. Dengan *bad-character rule* (atau pada salindia perkuliahan dikenal sebagai *Last Occurrence Function*), BM seringkali jauh lebih cepat daripada *brute-force* atau KMP pada teks alami berbahasa Inggris, karena rata-rata lompatan mendekati  $\Theta(n/m)$ . Selain itu, pada alfabet besar, BM semakin efisien sehingga cocok untuk CV yang umumnya berisi karakter ASCII standar. Meski demikian, dalam kasus terburuk BM dapat mencapai  $O(nm + A)$ , dengan  $A$  adalah ukuran alfabet. Dalam aplikasi ATS, BM dapat dipakai ketika karakteristik teks memungkinkan *mismatch* jarang terjadi di awal pola, sehingga loncatan jauh dapat sering dipakai.

Untuk kebutuhan *multi-keyword* dalam satu kali pemindaian, misalnya daftar skill atau posisi jabatan yang bisa mencapai puluhan atau ratusan istilah, Aho-Corasick (AC) menjadi kandidat utama yang terbaik untuk menangani kasus ini. AC membangun sebuah trie dari seluruh *pattern*, dilengkapi *failure links* yang mengarahkan ke *node* terdekat jika terjadi *mismatch*, sehingga seluruh kumpulan pola diproses dalam satu lintasan teks. Kompleksitasnya jauh lebih baik ketimbang memanggil KMP atau BM untuk setiap pola secara terpisah. Dengan demikian, AC sangat sesuai jika sistem diharuskan memeriksa puluhan keyword dalam sekali eksekusi, menghemat waktu dan memori dibandingkan looping exact match.

Meski exact match memadai untuk ejaan yang tepat, CV yang diunggah dapat mengandung typo atau variasi format. Untuk itu, Levenshtein Distance digunakan sebagai metode *fuzzy matching*. Dengan menghitung jarak edit, operasi penyisipan, penghapusan, dan penggantian, antara pola dan setiap substring teks sepanjang pola, sistem dapat mendeteksi kecocokan meski terdapat perbedaan kecil. *Early-exit pruning* dapat diterapkan untuk menghentikan perhitungan DP bila nilai minimum baris sudah melebihi ambang batas, menjaga agar kompleksitas rata-rata tidak terlalu membengkak.

### **3.3 Fitur Fungsional dan arsitektur aplikasi**

#### **3.3.1 Fitur pemilihan algoritma pencocokkan string**

Aplikasi memungkinkan pengguna untuk memilih algoritma pencocokkan *string* yang ingin digunakan. Terdapat tiga algoritma, yakni Knuth-Morris-Pratt, Boyer-Moore, dan Aho-Corasick yang tersedia pada *dropdown* di fitur utama.

#### **3.3.2 Fitur pemilihan jumlah curriculum vitae yang ingin dicari**

Aplikasi memungkinkan pengguna untuk memilih seberapa banyak *curriculum vitae* (cv) yang ingin dicari. Selain itu, aplikasi juga mengusahakan agar menampilkan cv dengan jumlah *match* terbanyak dengan algoritma pencocokkan *string* yang dipilih dan dengan bantuan *fuzzy search* menggunakan Jarak Levenshtein. Untuk pemilihan jumlah *curriculum vitae*, maksimum jumlah yang bisa dipilih adalah sebanyak 100 cv.

#### **3.3.3 Fitur pencarian single query atau multiple query**

Aplikasi memungkinkan pengguna untuk mencari *pattern* yang ingin dicari pada *database*. Pencarian bisa secara tunggal maupun jamak. Untuk pencarian tunggal, pengguna cukup untuk memasukkan sebuah kueri pada *search bar* yang ada pada tampilan utama *search* (contoh: “chef”), sedangkan untuk pencarian jamak, pengguna bisa memisahkan kueri dengan tanda koma dilanjutkan dengan kueri selanjutnya (contoh: “chef, cui, sine”).

#### **3.3.4 Fitur menampilkan ringkasan dari applicant yang cocok**

Aplikasi dapat menampilkan ringkasan dari *curriculum vitae* yang dimiliki oleh *applicant* yang cocok dengan *pattern* dan kriteria yang diinginkan oleh pengguna. Informasi ringkasan tersebut meliputi nama panjang (*full name*, yang termasuk *first name* dan *last name*), tanggal lahir (*birth date*), alamat (*address*), nomor telepon (*phone number*), keahlian (*skills*), riwayat pengalaman pekerjaan (*job history*), dan riwayat pendidikan (*education*).

#### **3.3.5 Fitur menampilkan cv dari applicant yang cocok**

Aplikasi dapat menampilkan *curriculum vitae* yang dimiliki oleh *applicant* yang cocok dengan *pattern* dan kriteria yang diinginkan oleh pengguna. Informasi *curriculum vitae* akan ditampilkan dengan PyQT Web Engine yang menunjang tampilan *web engine* pada GUI, sehingga pengguna dapat secara fleksibel melihat pdf dan juga dapat melakukan *zoom in* maupun *zoom out*.

#### **3.3.6 Arsitektur Aplikasi**

Arsitektur dari aplikasi yang kami ciptakan adalah sebagai berikut

1. Tampilan utama
  - 1.1. Judul & Logo Aplikasi
  - 1.4. Area tombol
    - 1.4.1. Tombol mulai pencarian (*search*)
    - 1.4.2. Tombol tentang program (*about*)
    - 1.4.3. Tombol keluar (*exit*)
2. Tampilan pencarian
  - 2.1. *Dropdown* algoritma pencocokkan *string*
  - 2.2. Tombol *input* data jumlah cv yang akan dicari
  - 2.3. *Search bar input* dari *pattern* yang akan dicari
  - 2.4 Tombol mulai pencarian (*search*)
3. Tampilan hasil pencarian
  - 3.1. Waktu pencarian
  - 3.2. Jumlah *pattern match* per cv
  - 3.3. *Summary* dari *matched cv*
  - 3.4 Data cv dari *matched cv*
  - 3.5 Jumlah cv yang ditemukan
4. Tampilan tentang aplikasi (*about*)
  - 4.1. Informasi tentang aplikasi yang telah dibuat

### 3.4 Ilustrasi Kasus

Sebagai ilustrasi kasus, kami akan melakukan pencocokan *pattern* dari file pdf yang menghasilkan teks yang telah diekstrak dan disimpan pada memori. Langkah pertama sebelum melakukan pencocokan *string* adalah melakukan konversi file pdf menjadi bentuk teks untuk *pattern matching* dan bentuk teks untuk regex. Perlakuan ekstraksi untuk *pattern matching* dan regex berbeda, karena *pattern matching* melakukan *exact match* dengan hasil ekstraksi, dan regex mungkin untuk melakukan konsiderasi pada *white space* ataupun enter (“\n”), sehingga proses ekstraksinya pun berbeda.

Untuk *pattern matching*, kami melakukan konversi dengan cara membaca teks yang didapatkan dari pdf baris per baris, dan menulisnya secara sekuensial ke memori dengan mengubahnya menjadi *lower case* dan masing-masing kata dipisahkan oleh spasi sampai baris habis terbaca (karakter non-ascii juga diabaikan). Sedangkan untuk regex, kami melakukan ekstraksi per baris, dan menuliskannya ke memori juga per baris (tidak sekuensial), sehingga mungkin adanya *new space* atau *new line*. Berikut adalah contoh ilustrasi hasil dari ekstraksi yang dilakukan oleh aplikasi.

Cuplikan PDF:

FOOD PREP CHEF  
Skills

- Highly skilled in cooking and preparing a variety of cuisines
- Inborn ability to explore new cooking avenues
- Thorough understanding of sanitation needs of the kitchen
- Operate kitchen equipment such as ovens and grills for cooking purposes
- Maintain knowledge of all recipes so that the Head Chef's place can be filled in effectively in case of absenteeism

Summary

- Exceptional culinary insight.
- Knowledge of standard food preparation
- Ability to work in a high volume environment
- Chef in preparing exceptional meals
- Motivated food serving professional with 5+ years food and beverage experience in casual and fine dining.

Tabel 1 Hasil ekstraksi dari pdf

Hasil ekstraksi <i>pattern matching</i>	Hasil ekstraksi regex
food prep chef skills highly skilled in cooking and preparing a variety of cuisines inborn ability to explore new cooking avenues thorough understanding of sanitation needs of the kitchen operate kitchen equipment such as ovens and grills for cooking purposes maintain knowledge of all recipes so that the head chef's place can be filled in effectively in case of absenteeism summary exceptional culinary insight. knowledge of standard food preparation ability to work in a high volume environment chef in preparing exceptional meals motivated food serving professional with 5+ years food and beverage experience in casual and fine dining.	FOOD PREP CHEF Skills  Highly skilled in cooking and preparing a variety of cuisines Inborn ability to explore new cooking avenues Thorough understanding of sanitation needs of the kitchen Operate kitchen equipment such as ovens and grills for cooking purposes Maintain knowledge of all recipes so that the Head Chef's place can be filled in effectively in case of absenteeism  Summary  Exceptional culinary insight. Knowledge of standard food preparation Ability to work in a high volume environment Chef in preparing exceptional meals Motivated food serving professional with 5+ years food and beverage experience in casual and fine dining.

Dengan file pdf yang telah diekstrak menjadi dua bentuk teks yang berbeda, selanjutnya adalah melakukan pencocokan *string* dengan algoritma

Knuth-Morris-Pratt, Boyer-Moore, atau Aho-Corasick. Ada beberapa skema yang bisa terjadi. Pertama, jika pengguna melakukan pencarian *pattern* tunggal, dan ada cv yang *match* dengan *pattern* tersebut, maka hasil akan ditampilkan dengan urutan *descending* (dari *match* terbanyak ke *match* tersedikit). Kedua, jika pengguna melakukan pencarian *pattern* tunggal, dan tidak ada cv yang *match* dengan *pattern* tersebut, maka hasil akan ditampilkan sesuai dengan hasil *fuzzy search* dengan algoritma Jarak Levenshtein dan diurutkan *descending* (dari *match* terbanyak ke *match* tersedikit). Ketiga, jika pengguna mencari *n* data, dan *pattern* yang *match* tidak sampai *n*, maka data yang ditampilkan adalah data *match* yang diurut secara *descending*, yang selanjutnya data hasil *fuzzy search* ditampilkan dan diurut secara *descending*. Keempat, untuk *multiple pattern, handling* aplikasi mirip dengan *single pattern*.

Selanjutnya, regex digunakan untuk menampilkan *summary* dari *matched cv* yang ditampilkan kepada pengguna. Caranya adalah, aplikasi akan mencari *common word* yang mungkin muncul pada pdf dengan menggunakan regex. Contoh dari *possible regex pattern* yang mungkin muncul adalah “SUMMARY|Summary|PROFILE|Profile|OBJECTIVE|Objective|ABOUT|About|PERSONAL\s+STATEMENT|Personal\s+Statement|OVERVIEW|Overview” dan lainnya. Jika *match*, maka aplikasi akan menampilkan hasilnya pada *cv summary*.

## **BAB IV**

### **IMPLEMENTASI DAN PENGUJIAN**

#### **4.1 Spesifikasi Teknis Program**

##### **4.1.1 Struktur Data**

Struktur data utama yang dipakai oleh aplikasi dapat dilihat di bawah ini.

1. TrieNode (pada file `aho_corasick.py`), merupakan struktur data fundamental untuk implementasi algoritma Aho-Corasick yang digunakan dalam pencarian *multi-pattern*. Struktur ini dirancang sebagai *node* dalam pohon Trie yang menyimpan *children* sebagai *dictionary* untuk *node-node* anak, *is\_end\_of\_pattern* sebagai boolean penanda akhir *pattern*, *pattern* untuk menyimpan *string pattern* itu sendiri, *failure* sebagai *pointer* ke *node failure link* untuk optimasi pencarian, dan *output* sebagai *list* yang berisi semua *pattern* yang berakhir di *node* tersebut.
2. DatabaseConnection (pada file `models.py`), adalah struktur data yang bertanggung jawab mengelola koneksi ke *database MySQL*. Kelas ini mengenkapsulasi parameter koneksi dalam bentuk *dictionary* yang berisi informasi *host*, *port*, *database name*, *username*, dan *password*, serta menyimpan objek koneksi MySQL yang aktual. Struktur ini menyediakan *method* untuk membuka dan menutup koneksi, serta mengeksekusi *query*.
3. DatabaseManager (pada file `models.py`), adalah struktur data utama yang mengkoordinasikan semua operasi *database* dalam aplikasi. Sebagai *singleton instance*, struktur ini mengelola objek `DatabaseConnection`, `ApplicantProfile`, `ApplicationDetail`, dan `AutoDecryptHelper` sebagai komponen-komponen terintegrasi. `DatabaseManager` menyediakan *high-level methods* untuk operasi kompleks seperti `get_data_by_applicant_id` yang menggabungkan data profil dengan detail aplikasi, `get_all_applicants_data` untuk mengambil semua data pelamar dengan relasi lengkap, serta berbagai fungsi pencarian berdasarkan nama, role, CV path, dan kriteria lainnya. Struktur ini juga menangani *automatic decryption* untuk data sensitif dan menyediakan *advanced search capabilities* yang memungkinkan pencarian berdasarkan *multiple criteria* secara bersamaan.
4. `ApplicantProfile` dan `ApplicationDetail` (pada file [`models.py`](#)), adalah dua struktur data yang bekerja sama untuk mengelola informasi lengkap tentang pelamar kerja. `ApplicantProfile` menyimpan data personal seperti nama depan, nama belakang, tanggal lahir, alamat, dan

nomor telepon, sementara ApplicationDetail menyimpan informasi spesifik aplikasi seperti posisi yang dilamar dan path ke file CV. Kedua struktur ini menyediakan operasi CRUD (Create, Read, Update, Delete) lengkap dan dirancang untuk bekerja dengan sistem enkripsi data. Relasi antara keduanya memungkinkan satu pelamar memiliki *multiple application* untuk posisi yang berbeda.

5. KeywordSearcher (pada file searcher.py), adalah struktur data utama yang bertindak sebagai *unified interface* untuk berbagai algoritma pencarian *keyword*. Kelas ini dirancang dengan *pattern strategy* yang memungkinkan penggunaan algoritma pencarian yang berbeda (Boyer-Moore, KMP, Aho-Corasick, atau Levenshtein) melalui satu *interface* yang konsisten. Struktur ini menyimpan *algorithm object*, *flag case\_sensitive* untuk kontrol sensitivitas huruf, dan *whole\_word flag* untuk pencarian kata utuh. Desain ini memberikan fleksibilitas tinggi dalam pemilihan algoritma pencarian sesuai kebutuhan, sambil menyediakan fitur tambahan seperti normalisasi *case* dan *filtering word boundary*.
6. CVGrouper (pada file cv\_grouper.py), adalah struktur data kompleks yang bertanggung jawab untuk menganalisis dan mengelompokkan konten CV ke dalam *section-section* yang terstruktur. Kelas ini menyimpan *section\_patterns* sebagai *dictionary* yang berisi *pattern regex* untuk mengenali berbagai *section* CV seperti summary, skills, education, dan experience. Struktur ini mengimplementasikan algoritma *parsing* untuk mengekstrak informasi dari teks CV yang tidak terstruktur menjadi data terstruktur yang dapat diproses lebih lanjut.
7. PDFExtractor (pada file extractor.py), adalah struktur data yang menangani proses ekstraksi teks dari file PDF CV dan konversinya ke dalam format yang dapat diproses oleh sistem. Kelas ini menyimpan *data\_folder* sebagai *Path object* yang menunjukkan lokasi folder berisi file PDF, dan *extracted\_data* sebagai *dictionary* yang menyimpan hasil ekstraksi dalam dua format: *regex\_format* untuk pemrosesan dengan *regular expression* dan *pattern\_matching* untuk algoritma pencarian *string*. Struktur ini menggunakan *library* PyMuPDF untuk ekstraksi teks dan mengimplementasikan berbagai teknik *preprocessing* seperti normalisasi *spacing*, pembersihan karakter non-ASCII, dan *formatting* untuk meningkatkan akurasi pencarian.
8. SearchPage State (pada file search\_page.py), adalah struktur data yang mengelola *state* dan komunikasi dalam komponen GUI pencarian. Struktur ini menggunakan PyQt signals untuk komunikasi secara asinkron antar komponen, termasuk *result\_selected* untuk menangani pemilihan hasil pencarian, *cv\_selected* untuk pemilihan CV, dan *search\_completed* untuk notifikasi selesainya proses pencarian. Selain

- itu, struktur ini juga menyimpan `use_multiprocessing flag` yang mengontrol apakah pencarian dilakukan secara paralel.
9. `DatabaseConfig` (pada file `config.py`), adalah struktur data yang mengelola konfigurasi sistem secara terpusat, khususnya untuk koneksi `database` dan pengaturan enkripsi. Kelas ini membaca konfigurasi dari `environment variables` atau file `.env` dan menyimpannya dalam `field-field` seperti `host, port, database, user, password` untuk koneksi `database`, serta `encryption_password` untuk sistem enkripsi. Struktur ini mengimplementasikan *singleton pattern* untuk memastikan konsistensi konfigurasi di seluruh aplikasi dan menyediakan validasi untuk memastikan semua parameter yang diperlukan tersedia.
  10. `Search Algorithm Classes` (pada file `boyer_moore.py, kmp.py, levenshtein.py`), adalah kumpulan implementasi algoritma pencarian `string` yang masing-masing memiliki karakteristik dan *use case* berbeda. `BoyerMooreSearch` mengimplementasikan algoritma Boyer-Moore dengan tabel *last occurrence* untuk pencarian yang efisien pada teks panjang. `KMPSearch` menggunakan algoritma Knuth-Morris-Pratt dengan *border function* untuk pencarian dengan *preprocessing pattern* yang optimal. `LevenshteinSearch` mengimplementasikan algoritma *edit distance* untuk *fuzzy matching* dengan *tolerance parameter* yang dapat disesuaikan. Masing-masing algoritma dirancang untuk skenario pencarian yang spesifik, memberikan fleksibilitas dalam pemilihan strategi pencarian berdasarkan kebutuhan akurasi dan performa.
  11. `Protocol Interfaces` (pada file `search_abc.py, multisearch_protocol.py, fuzzysearch_protocol.py`), adalah sekumpulan *abstract base classes* dan *protocols* yang mendefinisikan kontrak untuk berbagai jenis algoritma pencarian. `StringSearchAlgorithm` mendefinisikan *interface* untuk algoritma pencarian *single pattern*, `MultiPatternSearchAlgorithm` untuk algoritma yang dapat mencari *multiple patterns* dalam satu *pass*, dan `FuzzySearchAlgorithm` untuk algoritma pencarian *approximate* dengan *threshold*.

#### **4.1.2 Fungsi dan Prosedur**

Beberapa fungsi dan prosedur yang digunakan pada algoritma utama adalah sebagai berikut.

Fungsi dan Prosedur	Deskripsi
<code>perform_search()</code>	Fungsi utama yang menginisiasi pencarian CV berdasarkan <i>keywords</i> .

	Menjalankan pencarian dalam <i>thread</i> terpisah untuk menjaga responsivitas GUI dan menampilkan hasil dalam bentuk <i>card</i> .
_perform_search()	Implementasi <i>core logic</i> pencarian yang melakukan <i>exact matching</i> menggunakan <i>multiprocessing</i> , dilanjutkan dengan <i>fuzzy matching</i> jika diperlukan. Mengembalikan hasil terstruktur dengan <i>timing information</i> .
search_exact_worker()	<i>Worker function</i> untuk pencarian <i>exact match</i> yang dijalankan secara paralel. Mengekstrak teks dari PDF dan melakukan pencarian menggunakan algoritma yang dipilih (BM/KMP/Aho-Corasick).
search_fuzzy_worker()	<i>Worker function</i> untuk pencarian <i>fuzzy match</i> menggunakan algoritma Levenshtein. Digunakan untuk mencari keywords yang tidak ditemukan dalam <i>exact search</i> dengan toleransi tertentu.
extract_single_pdf()	Fungsi untuk mengekstrak teks dari satu file PDF CV. Menghasilkan dua format: <i>regex_format</i> untuk pemrosesan regex dan <i>pattern_matching</i> untuk algoritma pencarian <i>string</i> .
extract_all_pdffs()	Memproses semua file PDF dalam folder data secara <i>batch</i> . Menampilkan <i>snippet</i> hasil ekstraksi dan menyimpan data untuk pemrosesan lebih lanjut.
group_cv_data()	Fungsi utama untuk mengelompokkan konten CV yang sudah diekstrak ke dalam struktur data terorganisir ( <i>summary</i> , <i>skills</i> , <i>jobs</i> , <i>education</i> ).
extract_cv_sections()	Menganalisis teks CV menggunakan <i>pattern regex</i> untuk mengidentifikasi dan memisahkan <i>section-section</i> seperti <i>summary</i> , <i>skills</i> , <i>education</i> , dan <i>experience</i> .
_parse_skills()	Memproses <i>section skills</i> dari CV dengan menghilangkan <i>bullet points</i> , memisahkan <i>skills</i> yang digabung dengan <i>delimiter</i> , dan menghapus

	duplicasi.
_parse_experience()	Menganalisis <i>section experience</i> untuk mengekstrak informasi pekerjaan dalam format terstruktur dengan <i>position</i> , <i>year</i> , dan <i>description</i> .
_parse_education()	Memproses <i>section education</i> untuk mengekstrak informasi pendidikan dalam format <i>major</i> , <i>institution</i> , dan <i>year</i> dengan berbagai <i>pattern recognition</i> .
get_data_by_applicant_id()	Mengambil data lengkap pelamar berdasarkan ID, menggabungkan profil personal dengan semua detail aplikasi, dan melakukan dekripsi otomatis jika diperlukan.
get_all_applicants_data()	Mengambil semua data pelamar dari <i>database</i> dengan relasi lengkap antara profil dan aplikasi. Mendukung <i>limiting</i> untuk performa dan <i>automatic decryption</i> .
search_applicants_by_name()	Melakukan pencarian pelamar berdasarkan nama dengan dekripsi data terlebih dahulu untuk memastikan akurasi pencarian pada data terenkripsi.
advanced_search()	Implementasi pencarian <i>multi-criteria</i> yang memungkinkan <i>filtering</i> berdasarkan nama, tanggal lahir, nomor telepon, alamat, role, dan CV path secara bersamaan.
search()	Fungsi <i>unified search interface</i> yang dapat menangani <i>exact matching</i> (StringSearchAlgorithm, MultiPatternSearchAlgorithm) atau <i>fuzzy matching</i> (FuzzySearchAlgorithm) dengan <i>preprocessing case</i> dan <i>word boundary</i> .
search_multi()	Fungsi untuk pencarian <i>multiple patterns</i> dalam satu pass menggunakan algoritma seperti Aho-Corasick. Mengembalikan <i>mapping pattern</i> ke list posisi kemunculan.
search_fuzzy()	Implementasi pencarian <i>approximate</i>

	<i>matching</i> menggunakan <i>Levenshtein distance</i> dengan <i>tolerance parameter</i> . Mengembalikan posisi dan jarak <i>edit</i> untuk setiap match.
build_trie()	Membangun struktur Trie dari <i>list patterns</i> untuk algoritma Aho-Corasick. Setiap <i>node</i> menyimpan informasi <i>children</i> , <i>end-of-pattern</i> , dan <i>output patterns</i> .
build_failure_function()	Membangun <i>failure links</i> untuk algoritma Aho-Corasick menggunakan BFS. Mengoptimalkan pencarian dengan menghindari <i>backtracking</i> yang tidak perlu.
compute_border_function()	Menghitung <i>border function</i> untuk algoritma KMP yang digunakan untuk menentukan <i>shift</i> yang optimal saat terjadi <i>mismatch</i> dalam <i>pattern matching</i> .
format_for_regex()	Memproses teks mentah PDF untuk format yang optimal untuk <i>regex processing</i> dengan <i>preserving structure</i> dan <i>proper spacing</i> antar <i>section</i> .
format_for_pattern_matching()	Mengkonversi teks PDF ke format <i>lowercase continuous</i> untuk algoritma <i>string matching</i> dengan menghilangkan karakter khusus dan <i>normalizing spacing</i> .
populateContent()	Mengisi konten halaman CV summary dengan data yang sudah diparse dan dikelompokkan. Membuat <i>layout</i> dinamis untuk menampilkan <i>skills</i> , <i>experience</i> , dan <i>education</i> .
create_result_card()	Membuat <i>widget card</i> untuk menampilkan hasil pencarian dalam GUI. Termasuk informasi jumlah <i>matches</i> , nama pelamar, dan tombol untuk melihat summary atau CV.
loadPDF()	Memuat dan menampilkan file PDF CV dalam QWebEngineView widget untuk melihat langsung dalam aplikasi tanpa membuka aplikasi eksternal.

### 4.1.3 Algoritma

Pseudocode untuk algoritma Knuth-Morris-Pratt

```
procedure ComputeBorderFunction(input pattern : string,
output border : array of integer)
{ Menghitung fungsi border untuk pattern
Masukan: pattern - string yang akan dicari
Luaran: border - array fungsi border }

Deklarasi:
    m : integer
    j, i : integer

Algoritma:
    m ← length(pattern)
    border[1] ← 0
    j ← 0
    i ← 1

    while i < m do
        if pattern[i] = pattern[j] then
            j ← j + 1
            border[i] ← j
            i ← i + 1
        else
            if j ≠ 0 then
                j ← border[j - 1]
            else
                border[i] ← 0
                i ← i + 1
            endif
        endif
    endwhile

procedure KMPSearch(input text : string, input pattern :
string, output positions : array of integer)
{ Mencari semua kemunculan pattern dalam text menggunakan
algoritma KMP
Masukan: text - string tempat pencarian, pattern - string
yang dicari
Luaran: positions - array posisi kemunculan pattern }

Deklarasi:
    n, m : integer
    border : array of integer
    i, j : integer
    count : integer

Algoritma:
    n ← length(text)
    m ← length(pattern)
```

```

if m = 0 then
    { Pattern kosong cocok di semua posisi }
    for i ← 0 to n do
        positions[i] ← i
    endfor
    return
endif

ComputeBorderFunction(pattern, border)

i ← 0
j ← 0
count ← 0

while i < n do
    if text[i] = pattern[j] then
        i ← i + 1
        j ← j + 1
    endif

    if j = m then
        positions[count] ← i - j
        count ← count + 1
        j ← border[j - 1]
    else if i < n and pattern[j] ≠ text[i] then
        if j ≠ 0 then
            j ← border[j - 1]
        else
            i ← i + 1
        endif
    endif
endwhile

```

### Pseudocode untuk algoritma Boyer-Moore

```

procedure BuildLastOccurrence(input pattern : string, input
text : string, output lastOcc : map)
{ Membangun tabel last occurrence untuk karakter-karakter
Masukan: pattern - string yang dicari, text - string
tempat pencarian
Luaran: lastOcc - map posisi terakhir setiap karakter }

Deklarasi:
    chars : set of character
    i : integer

Algoritma:
    chars ← union(set(text), set(pattern))

    { Inisialisasi semua karakter dengan -1 }
    for each c in chars do

```

```

        lastOcc[c] ← -1
    endfor

    { Update posisi terakhir setiap karakter dalam pattern
}

    for i ← 0 to length(pattern) - 1 do
        lastOcc[pattern[i]] ← i
    endfor

procedure BoyerMooreSearch(input text : string, input
pattern : string, output positions : array of integer)
{ Mencari semua kemunculan pattern dalam text menggunakan
algoritma Boyer-Moore
Masukan: text - string tempat pencarian, pattern - string
yang dicari
Luaran: positions - array posisi kemunculan pattern }

Deklarasi:
n, m : integer
lastOcc : map
shift, j : integer
count : integer

Algoritma:
n ← length(text)
m ← length(pattern)

if m = 0 then
    { Pattern kosong cocok di semua posisi }
    for i ← 0 to n do
        positions[i] ← i
    endfor
    return
endif

BuildLastOccurrence(pattern, text, lastOcc)

shift ← 0
count ← 0

while shift ≤ n - m do
    j ← m - 1

    { Scan dari kanan ke kiri }
    while j ≥ 0 and pattern[j] = text[shift + j] do
        j ← j - 1
    endwhile

    if j < 0 then
        { Pattern ditemukan }
        positions[count] ← shift
        count ← count + 1

        { Hitung shift berikutnya }

```

```

        nextIdx ← shift + m
        if nextIdx < n then
            shift ← shift + m - lastOcc[text[nextIdx]]
        else
            shift ← shift + 1
        endif
    else
        { Hitung skip berdasarkan bad character rule }
        k ← lastOcc[text[shift + j]]
        rawSkip ← j - k
        shift ← shift + max(1, rawSkip)
    endif
endwhile

```

### Pseudocode untuk algoritma Aho-Corasick

```

structure TrieNode:
    children : map of character to TrieNode
    isEndOfPattern : boolean
    pattern : string
    failure : pointer to TrieNode
    output : array of string

procedure BuildTrie(input patterns : array of string,
output root : TrieNode)
{ Membangun trie dari sekumpulan pattern
  Masukan: patterns - array pattern yang akan dicari
  Luaran: root - root node dari trie }

Deklarasi:
    current : pointer to TrieNode
    i, j : integer

Algoritma:
    root ← new TrieNode()

    for i ← 0 to length(patterns) - 1 do
        if patterns[i] = "" then
            continue
        endif

        current ← root
        for j ← 0 to length(patterns[i]) - 1 do
            char ← patterns[i][j]
            if char not in current.children then
                current.children[char] ← new TrieNode()
            endif
            current ← current.children[char]
        endfor

        current.isEndOfPattern ← true

```

```

        current.pattern ← patterns[i]
        append patterns[i] to current.output
    endfor

procedure BuildFailureFunction(input root : TrieNode)
{ Membangun failure links untuk automaton Aho-Corasick
  Masukan: root - root node dari trie }

Deklarasi:
queue : queue of TrieNode
current, child, failure : pointer to TrieNode
char : character

Algoritma:
{ Inisialisasi failure link untuk node level 1 }
for each child in root.children do
    child.failure ← root
    enqueue child to queue
endfor

{ Proses sisa node menggunakan BFS }
while queue not empty do
    current ← dequeue from queue

    for each (char, child) in current.children do
        enqueue child to queue

        failure ← current.failure
        while failure ≠ null and char not in
failure.children do
            failure ← failure.failure
        endwhile

        if failure ≠ null then
            child.failure ← failure.children[char]
            append all patterns from
            child.failure.output to child.output
        else
            child.failure ← root
        endif
    endfor
endwhile

procedure AhoCorasickSearch(input text : string, input
patterns : array of string, output results : map)
{ Mencari semua kemunculan multiple patterns dalam text
  menggunakan Aho-Corasick
  Masukan: text - string tempat pencarian, patterns - array
  pattern yang dicari
  Luaran: results - map pattern ke array posisi kemunculan
}

Deklarasi:

```

```

root, current : pointer to TrieNode
i : integer
char : character
pattern : string
startPos : integer

Algoritma:
if text = "" then
    return
endif

BuildTrie(patterns, root)
BuildFailureFunction(root)

{ Inisialisasi results }
for each pattern in patterns do
    results[pattern] ← empty array
endfor

current ← root

for i ← 0 to length(text) - 1 do
    char ← text[i]

    { Ikuti failure link sampai menemukan transisi yang
cocok }
    while current ≠ null and char not in
current.children do
        current ← current.failure
    endwhile

    if current = null then
        current ← root
        continue
    endif

    { Pindah ke node berikutnya }
    current ← current.children[char]

    { Jika ada output di node ini, catat semua pattern
yang ditemukan }
    if current.output not empty then
        for each pattern in current.output do
            startPos ← i - length(pattern) + 1
            append startPos to results[pattern]
        endfor
    endif
endfor

```

### Pseudocode untuk algoritma Jarak Levenshtein

```

procedure LevenshteinDistance(input s1 : string, input s2 :

```

```

string, input maxDist : integer, output distance : integer)
{ Menghitung Levenshtein distance dengan early termination
  Masukan: s1, s2 - string yang dibandingkan, maxDist -
  maksimum distance yang diizinkan
  Luaran: distance - Levenshtein distance }

Deklarasi:
  n, m : integer
  prev, curr : array of integer
  i, j : integer
  cost, minRow : integer

Algoritma:
  n ← length(s1)
  m ← length(s2)

  { Inisialisasi array untuk row sebelumnya }
  for j ← 0 to m do
    prev[j] ← j
  endfor

  for i ← 1 to n do
    curr[0] ← i
    minRow ← curr[0]

    for j ← 1 to m do
      if s1[i-1] = s2[j-1] then
        cost ← 0
      else
        cost ← 1
      endif

      curr[j] ← min(
        prev[j] + 1,           { deletion }
        curr[j-1] + 1,         { insertion }
        prev[j-1] + cost     { substitution }
      )

      minRow ← min(minRow, curr[j])
    endfor

    { Early termination jika semua nilai melebihi
maxDist }
    if minRow > maxDist then
      distance ← maxDist + 1
      return
    endif

    { Swap arrays }
    swap(prev, curr)
  endfor

  distance ← prev[m]

```

```

procedure FuzzySearch(input text : string, input patterns :
array of string, input tolerance : real, output results :
map)
{ Mencari pattern dengan fuzzy matching menggunakan
Levenshtein distance
    Masukan: text - string tempat pencarian, patterns - array
pattern, tolerance - toleransi relatif (0-1)
    Luaran: results - map pattern ke array (posisi, distance)
}

Deklarasi:
    pattern : string
    m, maxEdits : integer
    i, distance : integer
    window : string

Algoritma:
    for each pattern in patterns do
        m ← length(pattern)
        results[pattern] ← empty array

        if m = 0 then
            { Pattern kosong cocok di semua posisi dengan
distance 0 }
            for i ← 0 to length(text) do
                append (i, 0) to results[pattern]
            endfor
            continue
        endif

        maxEdits ← ceiling(m × tolerance)

        for i ← 0 to length(text) - m do
            window ← substring(text, i, m)
            LevenshteinDistance(window, pattern, maxEdits,
distance)

            if distance ≤ maxEdits then
                append (i, distance) to results[pattern]
            endif
        endfor
    endfor
}

```

## 4.2 Tata Cara Penggunaan Program

Terdapat beberapa tahapan pendahuluan yang perlu dilakukan sebelum program bisa dijalankan,

1. klona repositori dengan perintah berikut

```
git clone https://github.com/l0stplains/Tubes3\_TheRecruiter.git
```

2. Navigasi ke folder klona repository dengan perintah

```
cd ./Tubes3_TheRecruiter
```

3. buat file .env yang berisi contoh sebagai berikut

```
# Database Configuration  
DB_HOST=localhost  
DB_PORT=2025  
DB_NAME=ats_system  
DB_USER=gongyoo  
DB_PASSWORD=REDACTED  
  
# MySQL Root  
MYSQL_ROOT_PASSWORD=REDACTED  
  
# Encryption Configuration  
ENCRYPTION_PASSWORD=REDACTED
```

4. Mengaktifkan database melalui docker dengan perintah berikut

```
docker-compose up -d
```

5. Pastikan penggunaan python 3.8 dengan menjalankan perintah berikut

```
uv python install 3.8  
uv python pin 3.8
```

6. unduh pustaka untuk masing masing sistem operasi

```
# Windows  
uv pip install PyQtWebEngine  
  
# Linux  
uv add PyQtWebEngine
```

7. Letakkan file-file CV pada folder data dalam root

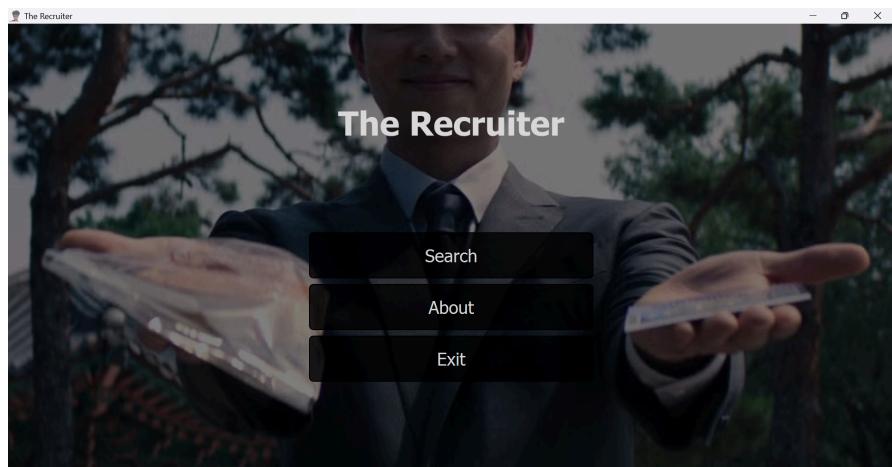
8. Jalankan skrip kode pengisi database dengan perintah berikut

```
uv run scripts/seeder.py
```

9. Jalankan program

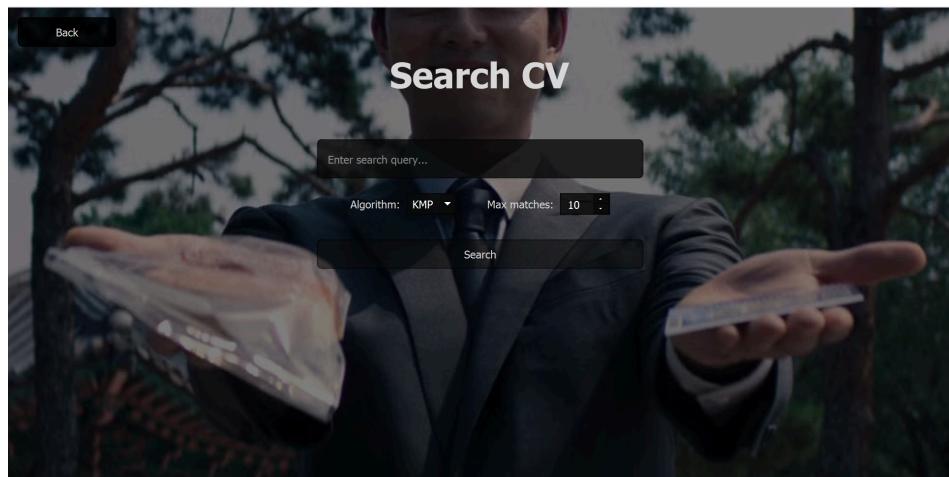
```
uv run -m src -d gui
```

Setelah program berhasil dijalankan, maka akan muncul layar berikut



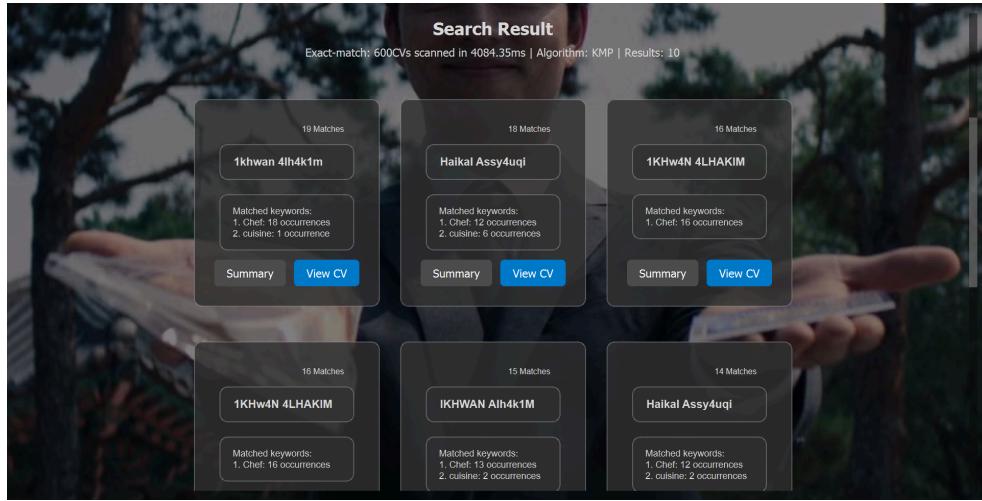
**Gambar 4.1** laman menu utama  
(Sumber: arsip penulis)

Pada laman ini, terdapat tiga tombol. tombol *search* akan membawa pengguna ke laman pencarian cv, tombol *About* akan membawa pengguna ke laman penjelasan program, dan tombol *exit* akan mengeluarkan pengguna dari program.



**Gambar 4.2** laman pencarian cv  
(Sumber: arsip penulis)

Pada laman ini, Pengguna dapat mengetik pada *text area* kata kunci yang ingin dicari pada cv. Kemudian pengguna memilih algoritma pencarian dan jumlah maksimal cv yang ingin ditampilkan ke layar laman. Terakhir, pengguna dapat menekan tombol *Search* untuk memulai pencarian cv.



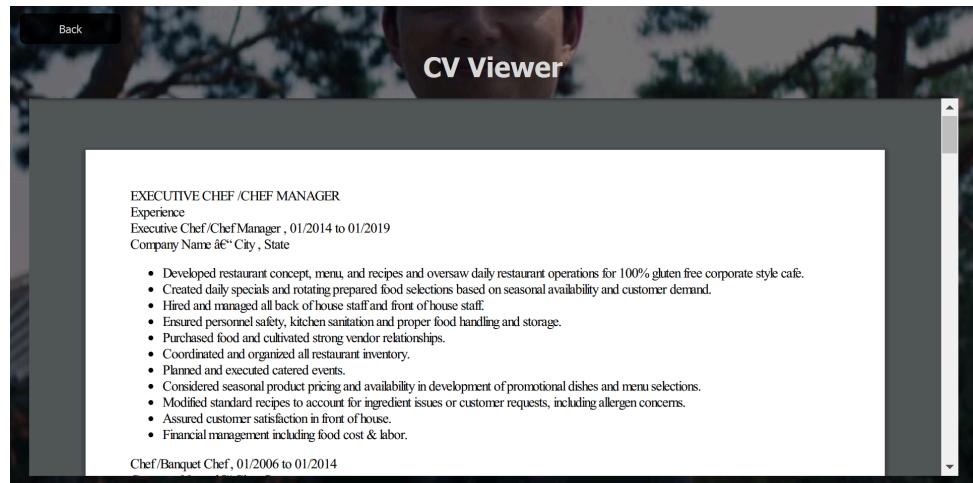
**Gambar 4.3** Hasil pencarian cv  
(Sumber: arsip penulis)

Hasil pencarian akan memunculkan ringkasan cv yang memiliki kata kunci yang dicari. Pada masing-masing ringkasan cv terdapat dua tombol. tombol *Summary* yang akan membawa pengguna ke laman ringkasan cv dan tombol *view cv* yang akan membawa pengguna untuk melihat file cv yang sebenarnya.



**Gambar 4.4** laman ringkasan cv  
(Sumber: arsip penulis)

Pada laman ini ditampilkan data ringkasan yang didapatkan dari file cv. empat hal utama yang ditampilkan pada laman ini, data personal, *skill*, edukasi dan, riwayat kerja pendaftar kerja.



**Gambar 4.5** laman cv

(Sumber: arsip penulis)

Pada laman ini ditampilkan cv asli yang dimiliki oleh pendaftar kerja.

### 4.3 Hasil Uji

#### 4.3.1 Pencarian kata kunci “Chef” dengan algoritma KMP

The screenshot shows a search interface for CVs. At the top, there is a search bar containing the word "Chef". Below it, there are dropdown menus for "Algorithm: KMP" and "Max matches: 10". A "Search" button is located below these controls. The main area is titled "Search Result" and displays the message "Exact-match: 600CVs scanned in 3866.58ms | Algorithm: KMP | Results: 10". Below this, three search results are shown in cards:

- 18 Matches: 1khwan 4lh4k1m  
Matched keywords: 1. Chef: 18 occurrences
- 16 Matches: 1KHw4N 4LHAKIM  
Matched keywords: 1. Chef: 16 occurrences
- 16 Matches: 1KHw4N 4LHAKIM  
Matched keywords: 1. Chef: 16 occurrences

At the bottom of the search results, there is a summary: "Exact match time: 3866 ms" and "Fuzzy match time: 0 ms".

#### 4.3.2 Pencarian kata kunci “Chef” dengan algoritma Boyer Moore

The screenshot shows a search interface for CVs. At the top, there is a search bar containing the word "Chef". Below it, a dropdown menu shows "Algorithm: BM" and a text input for "Max matches: 10". A "Search" button is present. The main area is titled "Search Result" and displays the message "Exact-match: 600CVs scanned in 4062.99ms | Algorithm: BM | Results: 10". Below this, three search results are shown in cards:

- 18 Matches: 1khwan 4lh4k1m. Matched keywords: 1. Chef: 18 occurrences.
- 16 Matches: 1KHw4N 4LHAKIM. Matched keywords: 1. Chef: 16 occurrences.
- 16 Matches: 1KHw4N 4LHAKIM. Matched keywords: 1. Chef: 16 occurrences.

Exact match time: 4062 ms  
Fuzzy match time: 0 ms

#### 4.3.3 Pencarian kata kunci “Chef” dengan algoritma Aho Corasick

The screenshot shows a search interface for CVs. At the top, there is a search bar containing the word "Chef". Below it, a dropdown menu shows "Algorithm: KMP" and a text input for "Max matches: 10". A "Search" button is present. The main area is titled "Search Result" and displays the message "Exact-match: 600CVs scanned in 3866.59ms | Algorithm: KMP | Results: 10". Below this, three search results are shown in cards:

- 18 Matches: 1khwan 4lh4k1m. Matched keywords: 1. Chef: 18 occurrences.
- 16 Matches: 1KHw4N 4LHAKIM. Matched keywords: 1. Chef: 16 occurrences.
- 16 Matches: 1KHw4N 4LHAKIM. Matched keywords: 1. Chef: 16 occurrences.

Exact match time: 4832 ms  
Fuzzy match time: 15478 ms

#### 4.3.4 Pencarian kata kunci “Ghana” dengan algoritma KMP

**Search CV**

Ghana

Algorithm: KMP ▾ Max matches: 10 :

Search

**Search Result**

Exact-match: 600CVs scanned in 4532.82ms | Fuzzy-match: 598CVs scanned in 19103.53ms | Algorithm: KMP | Results: 10

Matches	Keywords
1 Matches	1khw4n 4IH4k1m
Matched keywords:	1. Ghana: 1 occurrence

Matches	Keywords
1 Matches	1khw4n 4IH4k1m
Matched keywords:	1. Ghana: 1 occurrence

Matches	Keywords
11 Matches	Ari3L HerfR150n
Matched keywords:	Fuzzy matches: 1. Ghana: 11 fuzzy occurrences

Exact match time: 4532 ms  
Fuzzy match time: 19103 ms

#### 4.3.5 Pencarian kata kunci “Ghana” dengan algoritma Boyer Moore

**Search CV**

Ghana

Algorithm: BM ▾ Max matches: 10 :

Search

**Search Result**

Exact-match: 600CVs scanned in 4634.49ms | Fuzzy-match: 598CVs scanned in 16195.66ms | Algorithm: BM | Results: 10

Matches	Keywords
1 Matches	1khw4n 4IH4k1m
Matched keywords:	1. Ghana: 1 occurrence

Matches	Keywords
1 Matches	1khw4n 4IH4k1m
Matched keywords:	1. Ghana: 1 occurrence

Matches	Keywords
11 Matches	Ari3L HerfR150n
Matched keywords:	Fuzzy matches: 1. Ghana: 11 fuzzy occurrences

Exact match time: 4634 ms  
Fuzzy match time: 16195 ms

#### 4.3.6 Pencarian kata kunci “Ghana” dengan algoritma Aho Corasick

The screenshot shows the 'Search CV' application interface. At the top, there is a search bar containing the text 'Ghana'. Below the search bar are settings for 'Algorithm: AHO' and 'Max matches: 10'. A 'Search' button is located below these settings. The main area is titled 'Search Result' and displays the following information: 'Exact-match: 600CVs scanned in 4345.10ms | Fuzzy-match: 600CVs scanned in 17434.86ms | Algorithm: AHO | Results: 10'. Below this, three cards show search results: '11 Matches' (Ari3L HerfR150n), '4 Matches' (Mohammad NUGR4H4), and '3 Matches' (AR13L H3RFR1S0N).

Exact match time: 4345 ms  
Fuzzy match time: 17434 ms

#### 4.3.7 Pencarian kata kunci “Chef, cuisine, style” dengan algoritma KMP

The screenshot shows the 'Search CV' application interface. At the top, there is a search bar containing the text 'Chef, cuisine, style'. Below the search bar are settings for 'Algorithm: KMP' and 'Max matches: 10'. A 'Search' button is located below these settings. The main area is titled 'Search Result' and displays the following information: 'Exact-match: 600CVs scanned in 5902.79ms | Algorithm: KMP | Results: 10'. Below this, three cards show search results: '21 Matches' (1khwan 4lh4k1m) with 'Matched keywords: 1. Chef: 18 occurrences', '20 Matches' (Haikal Assy4uqi) with 'Matched keywords: 1. Chef: 12 occurrences', and '16 Matches' (1Khw4N 4LHAKIM) with 'Matched keywords: 1. Chef: 16 occurrences'.

Exact match time: 5902 ms  
Fuzzy match time: 0 ms

#### 4.3.8 Pencarian kata kunci “Chef, cuisine, style” dengan algoritma Boyer Moore

The screenshot shows the 'Search CV' application interface. At the top, there is a search bar containing the query 'Chef, cuisine, style'. Below the search bar, the algorithm is set to 'BM' and the maximum number of matches is set to 10. A 'Search' button is present. The results section is titled 'Search Result' and displays the following information: 'Exact-match: 600CVs scanned in 4276.96ms | Algorithm: BM | Results: 10'. Below this, three search results are shown in cards:

- 21 Matches: 1khwan 4lh4k1m  
Matched keywords:  
1. Chef: 18 occurrences  
2. cuisine: 1 occurrence
- 20 Matches: Haikal Assy4uqi  
Matched keywords:  
1. Chef: 12 occurrences  
2. cuisine: 6 occurrences
- 16 Matches: 1KHw4N 4LHAKIM  
Matched keywords:  
1. Chef: 16 occurrences

At the bottom of the results section, it says 'Exact match time: 4276 ms' and 'Fuzzy match time: 0 ms'.

#### 4.3.9 Pencarian kata kunci “Chef, cuisine, style” dengan algoritma Aho Corasick

The screenshot shows the 'Search CV' application interface. The search bar contains the query 'Chef, cuisine, style'. The algorithm is set to 'AHO' and the maximum number of matches is set to 10. A 'Search' button is present. The results section is titled 'Search Result' and displays the following information: 'Exact-match: 600CVs scanned in 5486.34ms | Algorithm: AHO | Results: 10'. Below this, three search results are shown in cards:

- 8 Matches: Haikal Assy4uqi  
Matched keywords:  
1. cuisine: 6 occurrences
- 3 Matches: ALaNd MuLiA  
Matched keywords:  
1. style: 3 occurrences
- 3 Matches: ALaNd MuLiA  
Matched keywords:  
1. style: 3 occurrences

At the bottom of the results section, it says 'Exact match time: 5486 ms' and 'Fuzzy match time: 0 ms'.

#### 4.3.10 Pencarian kata kunci “hahaha” dengan algoritma KMP

The screenshot shows a search interface for a CV database. The search term "hahaha" is entered in the search bar. The algorithm dropdown is set to "KMP" and the max matches is set to 10. The search results are displayed in three cards:

- 5 Matches**: M0H4MM4D NuGr4Ha  
Matched keywords:  
Fuzzy matches:  
1. hahaha: 5 fuzzy occurrences  
Buttons: Summary, View CV
- 2 Matches**: MoH4mM4d NugR4h4  
Matched keywords:  
Fuzzy matches:  
1. hahaha: 2 fuzzy occurrences  
Buttons: Summary, View CV
- 2 Matches**: MoH4mM4d NugR4h4  
Matched keywords:  
Fuzzy matches:  
1. hahaha: 2 fuzzy occurrences  
Buttons: Summary, View CV

Exact-match: 600CVs scanned in 1921.18ms | Fuzzy-match: 600CVs scanned in 4789.38ms | Algorithm: KMP | Results: 10

Exact match time: 1921.18 ms  
Fuzzy match time: 4789.38 ms

#### 4.3.11 Pencarian kata kunci “hahaha” dengan algoritma Boyer Moore

The screenshot shows a search interface for a CV database. The search term "hahaha" is entered in the search bar. The algorithm dropdown is set to "BM" and the max matches is set to 10. The search results are displayed in three cards:

- 5 Matches**: M0H4MM4D NuGr4Ha  
Matched keywords:  
Fuzzy matches:  
1. hahaha: 5 fuzzy occurrences  
Buttons: Summary, View CV
- 2 Matches**: MoH4mM4d NugR4h4  
Matched keywords:  
Fuzzy matches:  
1. hahaha: 2 fuzzy occurrences  
Buttons: Summary, View CV
- 2 Matches**: MoH4mM4d NugR4h4  
Matched keywords:  
Fuzzy matches:  
1. hahaha: 2 fuzzy occurrences  
Buttons: Summary, View CV

Exact-match: 600CVs scanned in 1928.17ms | Fuzzy-match: 600CVs scanned in 4832.71ms | Algorithm: BM | Results: 10

Exact match time: 1928.17 ms  
Fuzzy match time: 4832.71 ms

#### 4.3.12 Pencarian kata kunci “hahaha” dengan algoritma Aho Corasick

Exact match time: 1922.25 ms  
Fuzzy match time: 5083.99 ms

#### 4.4 Analisis Hasil Uji

Pada pengujian yang telah dilakukan sebelumnya, kita dapatkan waktu yang diperlukan berbagai algoritma untuk mencari suatu keyword serta hasil dari pencarian yang dilakukan. Telah kita pastikan bahwa hasil dari ketiga algoritma membawakan hasil pencarian yang sama hanya dengan waktu yang berbeda. Untuk pencarian kata kunci “Chef” yang merupakan kata yang akan relatif banyak ditemukan dalam suatu cv dibandingkan dengan kata kunci “ghana”, kita temukan bahwa algoritma KMP akan menyelesaikan pencarian dalam waktu yang paling singkat diikuti oleh algoritma Boyer Moore lalu algoritma Aho-Corasick. Sedangkan untuk kata kunci yang relatif lebih sedikit muncul pada cv, maka lebih cepat menggunakan algoritma Aho-Corasick diikuti dengan algoritma boyer-Moore lalu KMP.

Hal tersebut disebabkan oleh algoritma KMP lebih bersifat konsisten dibandingkan dengan kedua algoritma yang lain, secara teoretikal, KMP memiliki kompleksitas waktu sebesar  $O(n+m)$  baik untuk *worst-case* maupun *average-case*, di mana  $n$  adalah panjang teks dan  $m$  adalah panjang pola. Karena *pattern* “Chef” dan

“Ghana” adalah *pattern* yang cukup kecil, sehingga algoritma KMP memiliki keuntungan yang lebih.

Tetapi, hasil akan berbeda ketika sudah masuk ke *multiple pattern*. Karena algoritma Aho-Corasick memiliki mekanisme *failure node* dan beberapa konsep BFS, maka algoritma Aho-Corasick mungkin saja dapat menemukan *pattern* lain yang sedang dicari ketika sedang mencoba untuk melakukan *matching pattern* ke teksnya, menyebabkan waktu eksekusi algoritma Aho-Corasick lebih efisien dibandingkan dengan algoritma KMP, namun uniknya algoritma Boyer-Moore masih lebih cepat dibandingkan dengan Aho-Corasick. Hal ini disebabkan karena Boyer-Moore akan lebih diuntungkan jika *pattern* yang ingin di-*match* cukup beragam karena algoritma *mismatch*-nya, sedangkan jika dilihat, *pattern* pencarinya adalah “chef, cuisine, style” di mana ketiga *pattern* tersebut bersifat beragam satu sama lain.

Terakhir, adalah pengujian terhadap data dengan *pattern* yang kurang seragam, yakni “hahaha”. Hasil yang didapatkan adalah waktu eksekusi KMP adalah yang paling cepat dibandingkan dengan kedua algoritma yang lain. Hal ini sesuai dengan konsep KMP secara teoretikal, yakni akan diuntungkan jika *pattern* yang dicoba untuk *match* kurang beragam sehingga dia bisa mencocokkannya satu per-satu sesuai dengan *behaviour* ketika *mismatch*.

## **BAB V**

## **PENUTUP**

### **5.1 Kesimpulan**

Pada Tugas Besar 3 IF2211-Strategi Algoritma Semester 2 Tahun Ajaran 2024/2025, penulis diminta untuk membuat suatu aplikasi berbasis desktop yang menyelesaikan persoalan yang berkaitan dengan CV ATS-based dengan menggunakan algoritma pencocokkan string: Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan bonus yakni Aho-Corasick. Selain itu, penulis juga berhasil melakukan implementasi “error correction” pada algoritma pencocokkan *string* dengan menggunakan algoritma Jarak Levenshtein (*Levenshtein Distance*) dan juga melakukan implementasi pencocokkan pola pencarian kombinasi *string* dengan menggunakan *regular expression (regex)*.

Penulis berhasil membuat aplikasi yang dapat mencari informasi-informasi penting yang ada pada *curriculum vitae* dengan baik. Untuk menyelesaikan permasalahan tersebut, penulis menggunakan bahasa pemrograman Python dengan PyQt sebagai *Graphical User Interface (GUI)* dan juga menggunakan MySQL sebagai *Relational DataBase Management (RDBMS)* untuk menyimpan informasi dari *applicant* dan juga menyimpan beberapa informasi dari *curriculum vitae (cv)*.

### **5.2 Saran**

Pelaksanaan Tugas Besar IF2211-Strategi Algoritma Semester 2 Tahun Ajaran 2024/2025 merupakan pengalaman berharga dan juga berkesan bagi penulis. Dari pengalaman ini, selain belajar tentang algoritma dalam melakukan pencocokkan *string*, penulis juga belajar bagaimana cara untuk membangun aplikasi berbasis *desktop*. Penulis juga ingin memberikan beberapa saran kepada pembaca yang mungkin akan menghadapi tugas serupa di masa depan atau tertarik untuk mengembangkan topik ini lebih lanjut:

1. Jadwalkan dan buat roadmap yang jelas pada pengembangan aplikasi. Karena aplikasi yang ingin dibangun secara *scalability*-nya menengah sampai tinggi, maka hal yang bijak jika dibangun suatu jadwal dan roadmap terlebih dahulu agar proses penggerjaan lebih terarah dan juga terstruktur.
2. Komunikasi antar anggota tim. Jika pengembangan aplikasi dilakukan dengan tim, maka komunikasi antar anggota sangat penting untuk mengetahui progress antar anggota dan juga menghindari suatu konflik dalam pekerjaan yang sedang dikerjakan.
3. Eksplorasi algoritma pencocokkan *string*. Pencocokkan *string* bisa dieksplorasi lebih lanjut dengan menggunakan algoritma *Backward Nondeterministic Dawg Matching (BNDM)*, *Rabin-Karp*, atau algoritma lainnya. Penggunaan algoritma Knuth-Morris-Pratt, Boyer-Moore, dan

- Aho-Corasick hanya sebagai beberapa contoh dari algoritma pencocokkan *string*, dan masih bisa dikembangkan lebih lanjut.
4. Eksplorasi bahasa pemrograman. Bahasa pemrograman yang digunakan bisa dieksplorasi lebih lanjut, tidak hanya sebatas Python saja. Bahasa lain bisa digunakan agar aplikasi yang dikembangkan lebih luas. Sebagai contoh, penggunaan bahasa pemrograman Java dengan JavaFX yang mendukung pengembangan aplikasi berbasis *desktop*.

### 5.3 Refleksi

Ruang perbaikan dan pengembangan dalam membangun aplikasi berbasis *desktop* dapat difokuskan pada beberapa aspek. Pertama, manajemen dan perencanaan waktu adalah bagian utama dalam membangun sebuah aplikasi dengan waktu hanya tiga minggu. Kami menyadari bahwa manajemen dan perencanaan waktu yang lebih baik dan terstruktur dapat meningkatkan efisiensi penggerjaan dan juga menghindari tekanan waktu karena suatu *deadline* yang sudah dekat.

Selain itu, pemahaman terhadap spesifikasi yang diberikan juga harus diperhatikan. Memahami secara hati-hati dan menyeluruh tentang apa saja yang perlu dibangun, diminta, dan juga yang harus ada pada aplikasi. Hal ini dapat mencegah adanya risiko kesalahan pada pembuatan aplikasi dan juga aplikasi dapat berjalan sesuai dengan harapan. Selain itu, penempatan perhatian pada revisi spesifikasi juga harus selalu diperhatikan untuk menyesuaikan hal-hal yang perlu disesuaikan pada aplikasi.

Komunikasi tim yang berjalan dan baik juga harus dibangun dan bisa diperbaiki. Membuat sebuah ruang komunikasi untuk tim yang jelas, terbuka, dan inklusif di antara anggota tim untuk menghindari adanya miskomunikasi dan juga menyampaikan *progress update* dari hal yang telah dikerjakan oleh masing-masing. Diskusi yang diadakan secara reguler dan terjadwal terkait pembangunan dan pengembangan aplikasi juga dapat meningkatkan keterlibatan dan pemahaman untuk semua anggota tim.

Terakhir, evaluasi pada masing-masing anggota tim. Evaluasi dapat berbentuk umpan balik untuk antar anggota tim maupun dari asisten. Umpan balik ini dapat dimanfaatkan sebagai sarana pengembangan diri dari masing-masing anggota tim untuk perbaikan pada waktu selanjutnya dan juga sebagai media untuk terus berkembang. Selalu terbuka dan juga mau untuk mendengarkan saran serta komitmen untuk belajar dari setiap pengalaman juga dapat meningkatkan kinerja dan juga kompetensi yang dimiliki pada waktu selanjutnya.

## LAMPIRAN

### Github Repository

Program dapat diakses pada [https://github.com/l0stplains/Tubes3\\_TheRecruiter](https://github.com/l0stplains/Tubes3_TheRecruiter)

### Video

Video demonstrasi program dapat diakses pada ...

### Miscellaneous (Tabel Poin)

Tabel 2 Poin yang dikerjakan

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 summary CV applicant.	✓	
7	Aplikasi dapat menampilkan CV applicant secara keseluruhan.	✓	
8	Membuat laporan sesuai dengan spesifikasi.	✓	
9	[Bonus] Membuat bonus enkripsi data profil applicant.	✓	
10	[Bonus] Membuat bonus algoritma Aho-Corasick.	✓	
11	[Bonus] Membuat bonus video dan diunggah pada Youtube.		✓

## DAFTAR PUSTAKA

M. Rinaldi. Pencocokan String (*String/Pattern Matching*). Diakses pada 15 Juni 2025 dari [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)

National Institute of Standards and Technology. ADVANCED ENCRYPTION STANDARD (AES). Diakses pada 15 Juni 2025 dari <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>

D. Joan, R. Vincent. AES Proposal: Rijndael. Diakses pada 15 Juni 2025 dari <https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf>

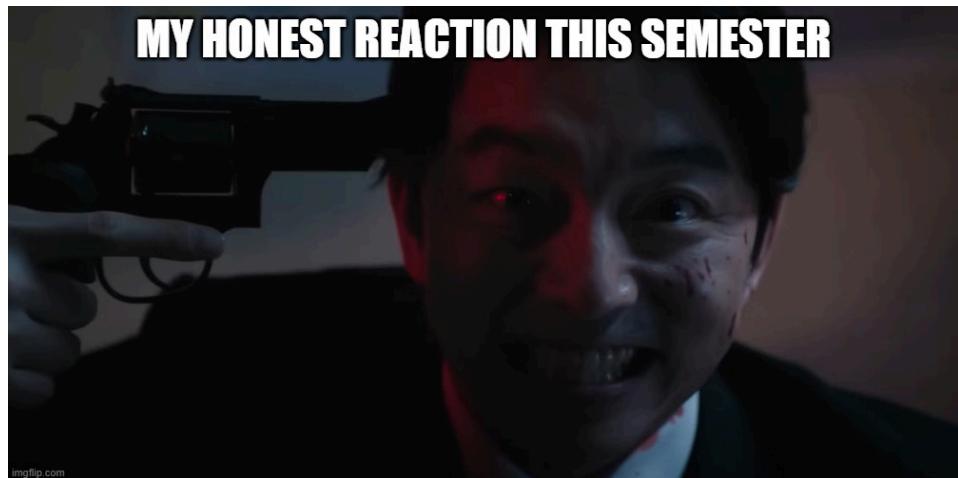
A.V. Aho, J.C. Margaret. String Matching: An Aid to Bibliographic Search. Diakses pada 15 Juni 2025 dari <https://cr.yp.to/bib/1975/aho.pdf>

H. Rishin, M. Debajyoti. Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. Diakses pada 15 Juni 2025 dari <https://arxiv.org/pdf/1101.1232>

L.K. Masayu. String Matching dengan Regular Expression. Diakses pada 15 Juni 2025 dari [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)

## AKHIR KATA

Sekian, *deliverables* dari kami, semoga AC, dan terima kasih para asisten dan dosen!



“I ENJOYED MY STAY 🤪 ”

-> Refki.

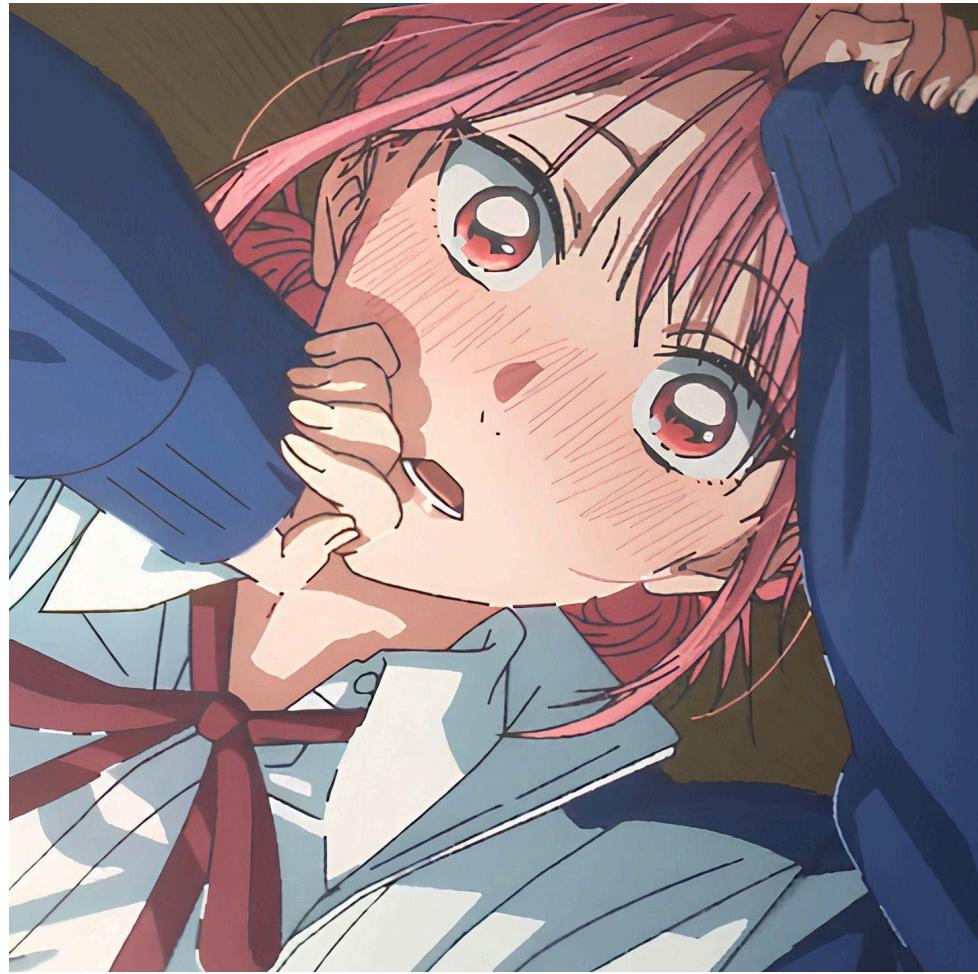
# My Alter Ego's Path to Greatness

[Home](#) > [Novel](#) > My Alter Ego's Path to Grea



“aku nak rekomen high mid. it aint peak but its alrite. its only 600 chap”

-> Jibril.



“If you really wanna go, i’ll be on your backseat”

Lullaby

-> Nayaka.

---