

LAPORAN TUGAS BESAR 3

IF2211 Strategi Algoritma

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



Disusun oleh:

Muh. Rusmin Nurwadin 13523068

Aryo Wisanggeni 13523100

Fityatul Haq Rosyidi 13523116

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

Daftar Isi

Daftar Isi.....	1
Daftar Gambar.....	3
BAB I.....	4
DESKRIPSI TUGAS.....	4
1.1 Deskripsi Tugas.....	4
1.2 Spesifikasi.....	4
BAB II.....	8
LANDASAN TEORI.....	8
2.1 Algoritma Knuth-Morris-Pratt.....	8
2.2 Algoritma Boyer-Moore.....	9
2.3 Algoritma Aho–Corasick.....	12
2.4 CVMatcher.....	13
BAB III.....	15
ANALISIS PEMECAHAN MASALAH.....	15
3.1 Langkah-langkah Pemecahan Masalah.....	15
3.1.1 Ekstraksi Teks dari Dokumen PDF.....	15
3.1.2 Algoritma Pencarian Exact Matching.....	15
3.1.3 Fuzzy Matching dengan Levenshtein Distance.....	15
3.1.4 Pengolahan Hasil dan Ekstraksi Informasi.....	15
3.2 Pemetaan Masalah.....	16
3.2.1 Pemetaan ke Algoritma KMP.....	16
3.2.2 Algoritma Boyer-Moore.....	16
3.2.3 Pemetaan ke Algoritma Aho-Corasick.....	17
3.2.4 Pemetaan ke Algoritma Levenshtein Distance.....	18
3.2.5 Pemetaan ke Regular Expression.....	18
3.3 Fitur fungsional dan Arsitektur Aplikasi.....	19
3.4 Contoh Ilustrasi Kasus.....	19
BAB IV.....	21
IMPLEMENTASI DAN PENGUJIAN.....	21
4.1 Implementasi Algoritma.....	21
4.1.1 Implementasi Algoritma KMP.....	21
4.1.2 Implementasi Algoritma BM.....	22
4.1.3 Implementasi Algoritma Aho-Corasick.....	22
4.1.4 Implementasi Algoritma Levenshtein Distance.....	25
4.1.5 Implementasi Regular Expression untuk Ekstraksi CV.....	25
4.1.6 Implementasi Enkripsi Data.....	29
4.2 Penggunaan Program.....	31
4.2.1 Setup Database MySQL.....	31
4.2.2 Setup Environment.....	32

4.2.3 Menjalankan Program.....	33
4.3 Struktur File.....	33
4.4 Hasil Pengujian.....	34
4.4.1 Algoritma KMP.....	34
4.4.2 Algoritma BM.....	38
4.4.3 Algoritma Aho-Corasick.....	42
4.4.4 Fuzzy Matching.....	46
4.4.5 Menampilkan summary CV applicant.....	48
4.4.6 Menampilkan CV applicant secara keseluruhan.....	49
4.4.7 Error Handling.....	50
4.4.8 Hasil Enkripsi.....	51
4.5 Analisis Hasil Pengujian.....	52
BAB V.....	53
KESIMPULAN DAN SARAN.....	53
5.1 Kesimpulan.....	53
5.2 Saran.....	53
5.3 Refleksi.....	53
LAMPIRAN.....	54
DAFTAR PUSTAKA.....	55

Daftar Gambar

Gambar 1. Ilustrasi Fungsi Border KMP.....	9
Gambar 2. Contoh Pencocokan Dengan KMP.....	10
Gambar 3. Contoh Kasus 1.....	11
Gambar 4. Contoh Kasus 2.....	11
Gambar 5. Contoh Kasus 3.....	11
Gambar 6. Contoh Last Occurrence Function.....	12
Gambar 7. Ilustrasi Penggunaan Algoritma.....	12
Gambar 8. Ilustrasi Algoritma (1).....	13
Gambar 9. Ilustrasi Algoritma (2).....	14
Gambar 10. Contoh Input.....	21

BAB I

DESKRIPSI TUGAS

1.1 Deskripsi Tugas

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

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

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

Di dalam Tugas Besar 3 ini, akan diimplementasikan 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.

1.2 Spesifikasi

Pada Tugas Besar ini, kami diminta untuk membuat sebuah sistem yang dapat melakukan pencocokan dan pencarian informasi pelamar kerja berdasarkan [dataset CV ATS Digital](#). Data yang digunakan merupakan gabungan **20 data pertama dari setiap *category/bidang*, yang telah terurut secara leksikografis** (i.e. 20 data dari *category* HR + 20 data dari *category* Designer, dst). Sistem harus memiliki fitur dengan detail sebagai berikut:

Spesifikasi Wajib

1. 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.

2. 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.
3. 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.
4. 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*.
5. 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.
6. 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**.
7. 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.

8. 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.
9. Pengguna aplikasi dapat **menentukan jumlah CV yang ditampilkan**. CV yang ditampilkan diurutkan mulai dari CV dengan jumlah kecocokan kata kunci terbanyak.
10. 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.
11. 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 Algoritma *Knuth-Morris-Pratt*

Algoritma Knuth-Morris-Pratt (KMP) merupakan metode pencocokan string yang bekerja secara berurutan dari kiri ke kanan, mirip dengan pendekatan Brute Force. Namun, perbedaannya terletak pada cara KMP melakukan pergeseran pola secara lebih efisien dengan memanfaatkan tabel prefix terbesar yang sekaligus merupakan suffix dari sebuah substring.

$$(k = j-1)$$

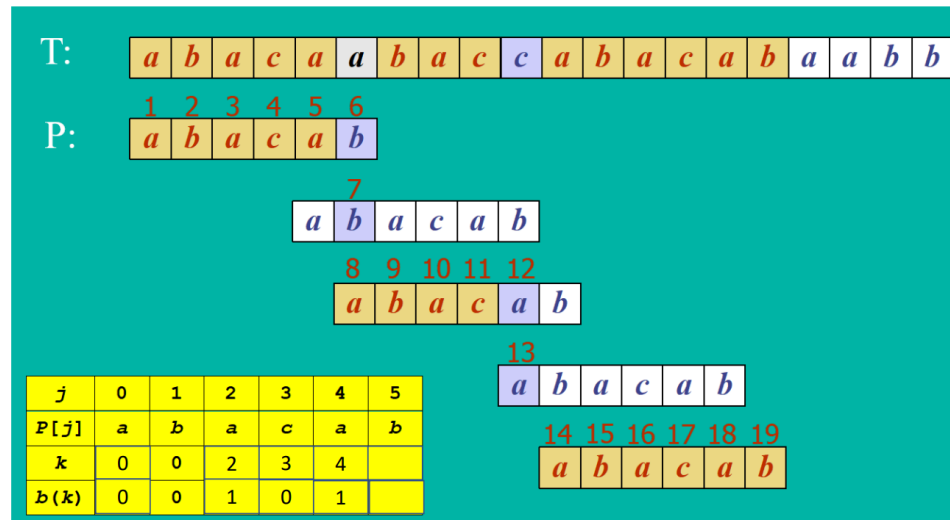
<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>P[j]</i>	a	b	a	b	a	b	a	b	c	a
<i>k</i>	0	1	2	3	4	5	6	7	8	
<i>b[k]</i>	0	0	1	2	3	4	5	6	0	

Gambar 1. Ilustrasi Fungsi Border KMP

Sumber :

[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)

Pada algoritma KMP, saat proses pencocokan berjalan dan terjadi ketidaksesuaian (*mismatch*) antara pola dan teks, pergeseran pola ditentukan berdasarkan border function. Border function ini digunakan untuk mencari prefix terpanjang yang juga berperan sebagai suffix pada substring yang telah cocok sebelumnya. Dengan demikian, KMP tidak perlu mengulang pencocokan dari awal, melainkan cukup melanjutkan dari posisi tertentu sesuai informasi dalam tabel tersebut.



Gambar 2. Contoh Pencocokan Dengan KMP

Sumber :

[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)

Selama proses pencocokan, KMP memang memiliki kemiripan dengan metode Brute Force, tetapi begitu ditemukan mismatch, pergeseran pola yang terjadi sebesar nilai yang ada pada tabel prefix-suffix. Dengan strategi ini, proses pencarian menjadi lebih efisien karena menghindari pemeriksaan ulang karakter yang sudah diketahui kecocokannya.

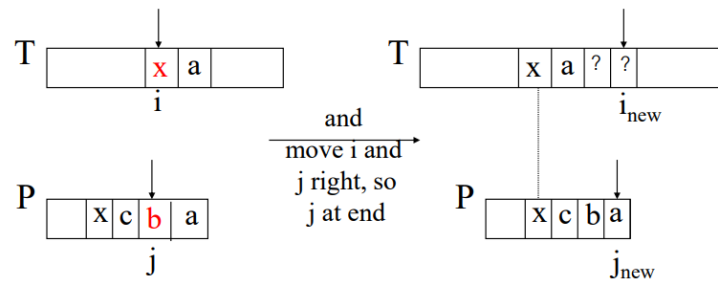
2.2 Algoritma Boyer-Moore

Algoritma Boyer-Moore merupakan salah satu metode pencocokan string yang dikenal sangat efisien, terutama ketika digunakan pada teks berukuran besar. Keunikan Boyer-Moore terletak pada dua teknik utamanya, yaitu looking-glass dan character-jump.

Teknik looking-glass dilakukan dengan mencocokkan pola (pattern) ke dalam teks mulai dari bagian paling kanan pola. Dengan kata lain, pencocokan karakter pada pola dimulai dari ujung belakang dan bergerak ke arah depan.

Sedangkan teknik character-jump digunakan ketika terjadi ketidaksesuaian antara karakter pola dan teks. Pada teknik ini, terdapat beberapa langkah yang dilakukan secara berurutan untuk menentukan pergeseran pola:

1. Jika karakter yang menyebabkan mismatch (x) ada di dalam pola, maka pola digeser ke kanan hingga posisi karakter x yang terakhir muncul dalam pola.

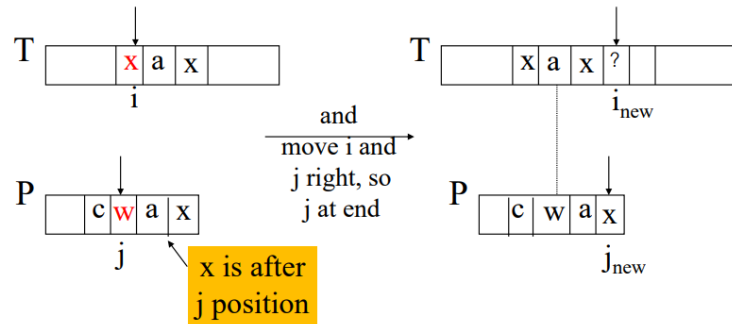


Gambar 3. Contoh Kasus 1

Sumber :

[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)

2. Jika karakter x memang ada dalam pola, namun tidak memungkinkan melakukan pergeseran seperti langkah pertama, maka pola digeser ke kanan sebanyak satu karakter saja.

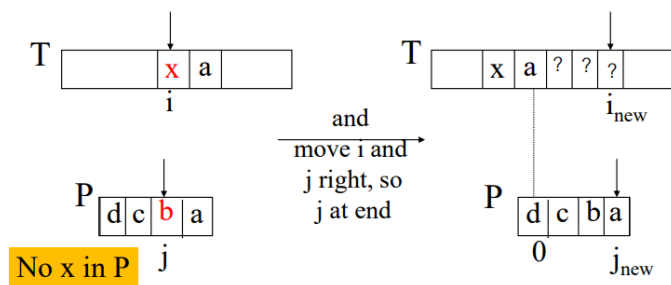


Gambar 4. Contoh Kasus 2

Sumber :

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

3. Jika kedua langkah sebelumnya tidak dapat dilakukan, maka pola digeser ke kanan sehingga karakter paling awal pola disejajarkan dengan karakter berikutnya pada teks.



Gambar 5. Contoh Kasus 3

Sumber :

[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)

Untuk membantu menentukan pergeseran yang optimal, algoritma Boyer-Moore membangun sebuah fungsi yang disebut Last Occurrence Function. Fungsi ini menghitung posisi indeks tertinggi dari setiap karakter dalam pola saat pertama kali pola diberikan. Jika suatu karakter tidak muncul dalam pola, maka nilainya -1. Informasi ini sangat membantu dalam mempercepat proses pencarian karena memungkinkan pola digeser lebih jauh saat terjadi mismatch.

L() Example

- $A = \{a, b, c, d\}$
- $P: "abacab"$

P	a	b	a	c	a	b
	0	1	2	3	4	5

x	a	b	c	d
$L(x)$	4	5	3	-1

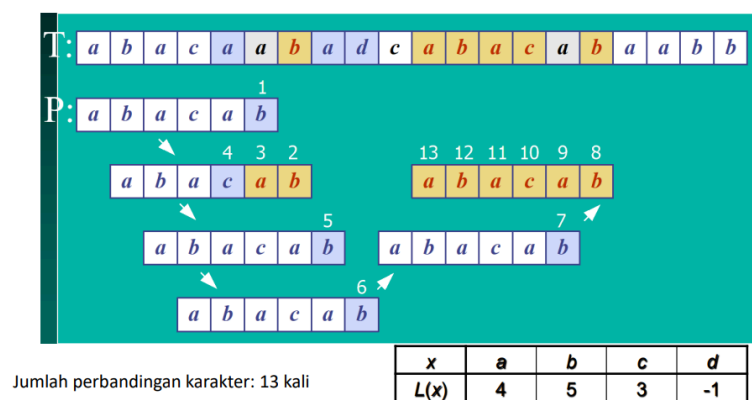
$L()$ stores indexes into $P[]$

Gambar 6. Contoh Last Occurrence Function

Sumber :

[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)

Boyer-Moore bekerja dengan memanfaatkan kedua teknik di atas secara bersamaan, serta didukung oleh fungsi last occurrence untuk menentukan pergeseran yang paling efisien. Dalam banyak kasus praktis, algoritma ini sangat cepat, terutama jika ukuran alphabet (A) besar. Namun, pada alphabet yang kecil, kecepatan Boyer-Moore cenderung menurun.



Gambar 7. Ilustrasi Penggunaan Algoritma

Sumber :

[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)

Dari sisi performa, kompleksitas waktu terburuk algoritma Boyer-Moore adalah $O(nm + A)$, di mana n adalah panjang teks, m adalah panjang pola, dan A adalah ukuran alphabet yang digunakan.

2.3 Algoritma Aho–Corasick

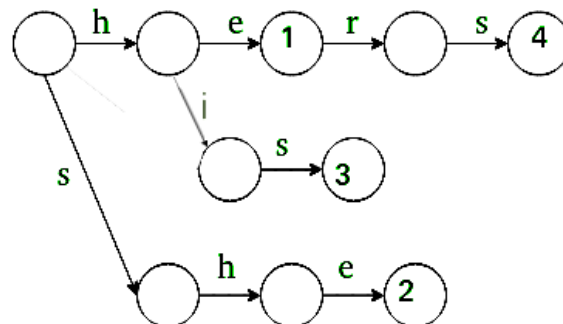
Algoritma Aho–Corasick adalah algoritma pencocokan string yang dirancang untuk menemukan banyak pola (pattern) sekaligus di dalam sebuah teks dengan efisiensi tinggi. Berbeda dengan algoritma pencocokan klasik yang bekerja satu pola dalam satu waktu, Aho–Corasick mampu mencari semua kemunculan dari sekumpulan pola sekaligus hanya dalam satu kali pemindaian teks.

Pada dasarnya, Aho–Corasick membangun struktur data berupa automaton (finite state machine) dari seluruh kumpulan pola yang ingin dicari. Automaton ini mirip dengan pohon trie, namun dilengkapi dengan fungsi tambahan yang disebut failure link. Setiap node pada trie mewakili satu state atau kondisi dalam proses pencocokan, sedangkan failure link menghubungkan node dengan node lain yang merupakan kemungkinan fallback ketika terjadi mismatch.

Tahapan utama dalam algoritma Aho–Corasick meliputi:

1. Membangun Trie dari semua pola: Semua pattern yang ingin dicari dimasukkan ke dalam struktur trie, di mana setiap cabang mewakili karakter dari pola tersebut.
2. Membangun Failure Link: Setelah trie terbentuk, algoritma akan menghitung failure link untuk setiap node, sehingga pencocokan dapat dilanjutkan dengan efisien saat terjadi mismatch, tanpa harus mengulang dari awal.
3. Pencarian di dalam teks: Teks akan dipindai satu per satu karakter. Pada setiap langkah, automaton berpindah state sesuai karakter yang dibaca. Jika ditemukan pola yang cocok, automaton langsung melaporkan posisi kecocokan di dalam teks.

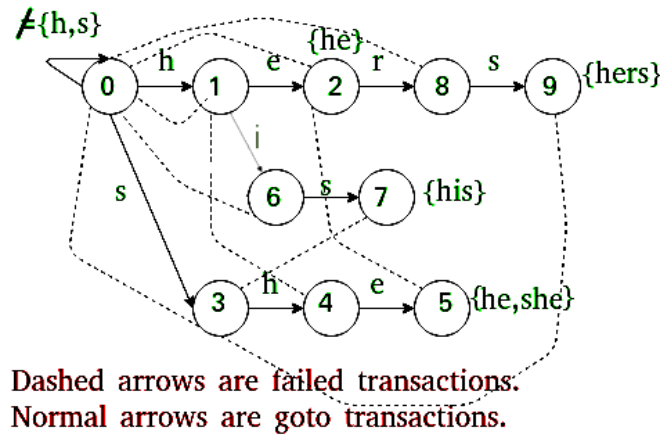
Trie for Arr[] = { he , she , his , hers }



Gambar 8. Ilustrasi Algoritma (1)

Sumber :

<https://www.geeksforgeeks.org/dsa/aho-corasick-algorithm-pattern-searching/>



Gambar 9. Ilustrasi Algoritma (2)

Sumber :

<https://www.geeksforgeeks.org/dsa/aho-corasick-algorithm-pattern-searching/>

Keunggulan utama Aho–Corasick adalah kemampuannya melakukan pencocokan banyak pola secara bersamaan dengan kompleksitas waktu yang efisien, yaitu $O(n + m + z)$, di mana n adalah panjang teks, m adalah total panjang semua pola, dan z adalah jumlah total kecocokan yang ditemukan.

2.4 CVMatcher

CVMatcher adalah aplikasi desktop yang dirancang khusus untuk membantu perusahaan dalam proses rekrutmen tenaga kerja secara lebih efisien dan akurat. Aplikasi ini hadir sebagai solusi modern untuk mengatasi tantangan yang sering dihadapi oleh tim HR dan rekruter dalam mengelola ratusan bahkan ribuan dokumen CV yang masuk setiap harinya. Dengan menggunakan teknologi canggih, CVMatcher mampu mengotomatisasi proses penyaringan dan pencocokan kandidat berdasarkan kriteria yang diinginkan perusahaan.

Salah satu keunggulan utama CVMatcher terletak pada kemampuannya dalam memproses dokumen CV dalam format PDF secara otomatis. Aplikasi ini dapat mengekstrak informasi penting dari setiap CV seperti data pribadi, pengalaman kerja, keahlian, dan riwayat pendidikan tanpa memerlukan input manual yang memakan waktu. Fitur pencarian yang disediakan memungkinkan pengguna untuk memasukkan kata kunci tertentu seperti nama skill, posisi, atau pengalaman yang dibutuhkan, kemudian sistem akan menampilkan daftar kandidat yang paling sesuai dengan kriteria tersebut.

Yang membuat CVMatcher istimewa adalah penggunaan algoritma pencarian string yang canggih seperti Knuth-Morris-Pratt dan Boyer-Moore untuk melakukan pencocokan kata kunci secara tepat dan cepat. Selain itu, aplikasi ini juga dilengkapi dengan fitur pencarian fuzzy menggunakan algoritma Levenshtein Distance yang dapat menoleransi kesalahan ketik minor, sehingga hasil pencarian tetap akurat meskipun terdapat perbedaan kecil dalam penulisan kata kunci. Hal ini sangat membantu tim HR yang mungkin tidak selalu mengetikkan kata kunci dengan ejaan yang persis sama.

CVMatcher tidak hanya sekadar mencari dan menampilkan daftar kandidat, tetapi juga menyediakan ringkasan profil yang komprehensif untuk setiap kandidat. Pengguna dapat melihat informasi detail seperti data kontak, pengalaman kerja, dan keahlian yang dimiliki dalam format yang mudah dibaca dan dipahami. Aplikasi ini juga memungkinkan pengguna untuk melihat dokumen CV asli secara langsung, memberikan fleksibilitas dalam proses evaluasi kandidat. Dengan antarmuka yang intuitif dan user-friendly, CVMatcher dirancang agar mudah digunakan bahkan oleh pengguna yang tidak memiliki latar belakang teknis, sehingga dapat mengoptimalkan proses rekrutmen dan membantu perusahaan menemukan talenta terbaik dengan lebih efisien.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

Tugas besar 3 Strategi Algoritma ini bertujuan untuk mencari kandidat pelamar kerja yang paling sesuai dengan kriteria yang diinginkan perusahaan melalui pencocokan kata kunci pada dokumen CV digital. Pendekatan yang digunakan melibatkan pemrosesan dokumen PDF menjadi teks yang dapat dianalisis secara algoritmik untuk menghasilkan rekomendasi kandidat terbaik.

3.1.1 Ekstraksi Teks dari Dokumen PDF

Proses pemecahan masalah dimulai dengan ekstraksi teks dari dokumen CV dalam format PDF menggunakan library PyMuPDF. Setiap dokumen CV yang tersimpan dalam database akan dikonversi menjadi string teks murni yang kemudian menjadi bahan untuk proses pencocokan. Tahap ini sangat penting karena dokumen PDF yang tidak terstruktur perlu diubah menjadi format yang dapat diproses oleh algoritma pencarian string.

3.1.2 Algoritma Pencarian Exact Matching

Tahap selanjutnya adalah implementasi algoritma pencarian string untuk mencocokkan kata kunci yang dimasukkan pengguna dengan konten CV yang telah diekstrak. Sistem menggunakan dua algoritma utama yaitu Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) untuk melakukan exact matching, di mana pencarian dilakukan secara presisi untuk menemukan kata kunci yang tepat sama. Pengguna dapat memilih algoritma mana yang ingin digunakan sesuai dengan preferensi atau kebutuhan performa.

3.1.3 Fuzzy Matching dengan Levenshtein Distance

Apabila proses exact matching tidak memberikan hasil yang memadai, sistem akan melanjutkan dengan fuzzy matching menggunakan algoritma Levenshtein Distance. Pendekatan ini memungkinkan sistem untuk tetap menemukan kandidat yang relevan meskipun terdapat perbedaan minor dalam penulisan atau kesalahan ketik pada kata kunci pencarian. Sistem akan mencari kecocokan dengan tingkat kemiripan di atas threshold tertentu yang telah ditentukan.

3.1.4 Pengolahan Hasil dan Ekstraksi Informasi

Setelah proses pencocokan selesai, sistem akan mengurutkan hasil berdasarkan jumlah kecocokan kata kunci dan menampilkan kandidat dengan skor tertinggi sesuai dengan jumlah yang diminta pengguna. Untuk melengkapi informasi, sistem juga menggunakan Regular Expression untuk mengekstrak informasi terstruktur dari CV seperti data kontak, pengalaman kerja, keahlian, dan riwayat pendidikan yang kemudian disajikan dalam format ringkasan yang mudah dipahami.

3.2 Pemetaan Masalah

3.2.1 Pemetaan ke Algoritma KMP

Masalah pencarian kandidat dipetakan menjadi permasalahan string matching, di mana kata kunci yang dimasukkan pengguna menjadi pattern yang harus dicari dalam text yang merupakan konten CV. Algoritma KMP dipetakan untuk menangani pencarian pattern dengan memanfaatkan tabel Longest Prefix Suffix (LPS) yang memungkinkan pencarian berlangsung dalam waktu linear dan efisien. Secara umum, ada dua tahap dalam algoritma KMP yaitu preprocessing dan pencarian.

Langkah-langkah implementasi algoritma KMP:

1. Pre-processing

- Buat array LPS (Longest Prefix Suffix) untuk pattern.
- Isi array LPS dengan panjang prefiks yang juga merupakan sufiks terpanjang untuk setiap posisi dalam pola.

2. Pencarian

- Bandingkan pola dengan teks utama dari awal.
- Jika karakter cocok, lanjutkan ke karakter berikutnya dari teks dan pola.
- Jika terjadi ketidakcocokan:
 - Jika posisi dalam pola bukan yang pertama, gunakan nilai dalam LPS Array untuk menentukan langkah selanjutnya dalam pola tanpa menggerakkan teks.
 - Jika posisi dalam pola adalah yang pertama, geser pola ke kanan satu posisi dalam teks dan lanjutkan perbandingan.
- Ulangi langkah ini sampai pola ditemukan atau teks habis.

3.2.2 Algoritma Boyer-Moore

Algoritma Boyer-Moore dipetakan untuk memberikan alternatif pencarian yang lebih cepat pada kasus tertentu dengan memanfaatkan teknik bad character dan good suffix untuk melompati karakter yang tidak perlu diperiksa. Algoritma BM digunakan untuk mencari pattern yang dihasilkan dari kata kunci dalam text yang dihasilkan dari ekstraksi CV. Secara umum, ada dua tahap dalam algoritma BM yaitu preprocessing dan pencarian.

Langkah-langkah implementasi algoritma Boyer-Moore:

1. Pre-processing

- Inisialisasi array last occurrence
- Isi array last occurrence dengan posisi terakhir kemunculan setiap karakter dalam pola

2. Pencarian

- Jika panjang pola lebih besar dari panjang teks, maka pencarian dihentikan dan mengembalikan false.
- Selama i lebih kecil dari panjang teks n , (i iterator pada teks dan j iterator pada pattern)
 - Jika karakter dalam pattern sama dengan karakter dalam teks:
 - Jika sudah mencapai awal pola, artinya pola ditemukan dalam teks, kembalikan true.
 - Jika belum mencapai awal pola, geser satu posisi ke kiri pada teks dan pola.
 - Jika karakter dalam pattern tidak sama dengan karakter dalam teks:
 - Temukan posisi terakhir karakter teks yang tidak cocok dalam pola menggunakan array `last_occurrence`.
 - Geser pola ke kanan dengan jumlah yang ditentukan oleh aturan Boyer-Moore, yaitu $m - \text{Math.Min}(j, 1 + k)$.
 - Kembalikan j ke posisi akhir pola.

3.2.3 Pemetaan ke Algoritma Aho-Corasick

Algoritma Aho-Corasick dipetakan sebagai solusi multi-pattern matching yang memungkinkan pencarian seluruh daftar kata kunci sekaligus dalam satu proses traversal teks. Hal ini sangat efisien dibandingkan pendekatan konvensional yang melakukan pencarian satu per satu untuk setiap kata kunci. Algoritma ini menggunakan struktur data trie yang dikombinasikan dengan failure function untuk melakukan pencarian yang optimal.

Langkah-langkah implementasi algoritma Aho-Corasick:

1. Konstruksi Trie

- a. Bangun trie dari semua pattern (kata kunci) yang akan dicari.
- b. Setiap node dalam trie merepresentasikan prefix dari salah satu pattern.

2. Konstruksi Failure Function

- a. Untuk setiap node, tentukan failure link yang menunjuk ke node lain yang merepresentasikan suffix terpanjang yang juga merupakan prefix dari pattern lain.
- b. Gunakan BFS (Breadth-First Search) untuk membangun failure function secara level by level.

3. Pencarian Multi-Pattern

- a. Traverse teks input karakter demi karakter.
- b. Untuk setiap karakter, ikuti transisi di trie jika memungkinkan.
- c. Jika tidak ada transisi yang valid, ikuti failure link sampai menemukan transisi yang valid atau kembali ke root.
- d. Catat semua pattern yang berakhir di node saat ini.

3.2.4 Pemetaan ke Algoritma Levenshtein Distance

Masalah toleransi kesalahan dalam pencarian dipetakan menjadi implementasi algoritma Levenshtein Distance yang mengukur jarak edit antar string. Hal ini memungkinkan sistem untuk tetap menemukan kecocokan meskipun terdapat perbedaan kecil dalam penulisan. Algoritma ini dihitung menggunakan dynamic programming untuk menentukan minimum edit distance antara kata kunci pencarian dan kata-kata dalam CV.

Langkah-langkah implementasi Levenshtein Distance:

1. Inisialisasi Matrix

- Buat matrix berukuran $(m+1) \times (n+1)$ dimana m adalah panjang string pertama dan n adalah panjang string kedua.
- Inisialisasi baris pertama dan kolom pertama dengan nilai 0, 1, 2, ... sebagai base case.

2. Perhitungan Dynamic Programming

Untuk setiap sel (i,j) , hitung nilai minimum dari, dengan ketiga operasi memiliki bobot yang sama, yaitu 1:

- a. $\text{Matrix}[i-1][j] + 1$ (deletion)
- b. $\text{Matrix}[i][j-1] + 1$ (insertion)
- c. $\text{Matrix}[i-1][j-1] + \text{cost}$ (substitution, dimana $\text{cost} = 0$ jika karakter sama, 1 jika berbeda)

3. Kalkulasi Similarity Score

- Setelah menghitung semua sel pada matriks, diperoleh Levenshtein distance antara dua string pada indeks $(m+1, n+1)$ pada distance matrix sebagai nilai similarity antar kedua string
- Semakin kecil Levenshtein distance, semakin similar kedua string, dengan $\text{distance} = 0$ mengartikan kedua string persis sama. Di program menggunakan levenshtein threshold 2, sehingga match yang Levenshtein distance = 2 atau kurang akan menjadi bagian dari hasil fuzzy search.

3.2.5 Pemetaan ke Regular Expression

Masalah ekstraksi informasi terstruktur dari dokumen tidak terstruktur dipetakan menjadi implementasi Regular Expression yang dapat mengenali pola-pola tertentu dalam teks seperti format email, nomor telepon, pengalaman kerja, dan keahlian. Pola regex dirancang untuk mengidentifikasi berbagai format penulisan informasi dalam CV yang beragam.

3.3 Fitur fungsional dan Arsitektur Aplikasi

CVMatcher memiliki beberapa fitur fungsional utama yang dirancang untuk memberikan pengalaman pencarian kandidat yang optimal. Fitur pencarian kata kunci memungkinkan pengguna untuk memasukkan satu atau lebih kata kunci yang dipisahkan dengan koma, yang kemudian akan dicocokkan dengan konten seluruh CV dalam database. Pengguna dapat memilih algoritma pencarian yang diinginkan antara KMP, Boyer-Moore, atau Aho-Corasick (untuk fitur bonus), memberikan fleksibilitas dalam pendekatan pencarian. Fitur pengaturan jumlah hasil memungkinkan pengguna untuk menentukan berapa banyak kandidat teratas yang ingin ditampilkan, dengan hasil yang diurutkan berdasarkan relevansi dan jumlah kecocokan kata kunci.

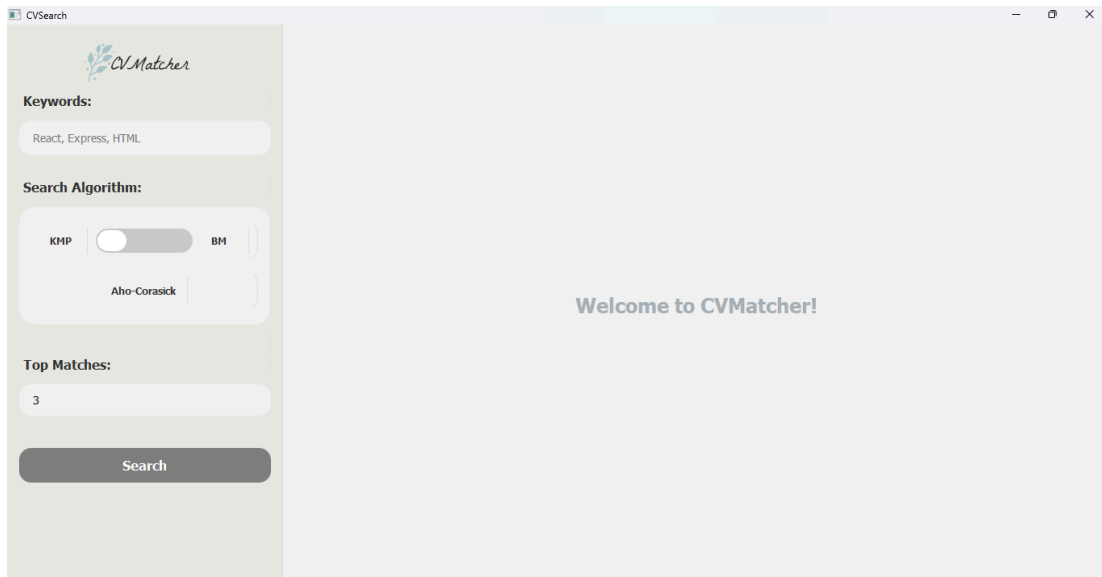
Sistem juga menyediakan fitur tampilan ringkasan kandidat yang menampilkan informasi penting seperti data kontak, pengalaman kerja, keahlian, dan riwayat pendidikan yang diekstrak secara otomatis menggunakan Regular Expression. Fitur view CV original memungkinkan pengguna untuk melihat dokumen CV asli dalam format PDF untuk evaluasi lebih detail. Selain itu, sistem menampilkan waktu eksekusi pencarian untuk memberikan transparansi tentang performa algoritma yang digunakan, serta informasi statistik tentang jenis pencocokan yang dilakukan (exact match vs fuzzy match).

Arsitektur aplikasi CVMatcher menggunakan pola Model-View-Controller (MVC) yang memisahkan tanggung jawab antar komponen sistem. Layer View menggunakan PyQt5 untuk menyediakan antarmuka pengguna yang intuitif dan responsif, memungkinkan interaksi yang mudah dalam input kata kunci, pemilihan algoritma, dan menampilkan hasil pencarian. Layer Controller bertindak sebagai mediator yang menangani logika bisnis, termasuk koordinasi antara proses pencarian, pengolahan hasil, dan komunikasi dengan database. Layer Model mengelola akses data melalui MySQL database yang menyimpan informasi profil pelamar dan detail aplikasi, serta menangani operasi file PDF untuk ekstraksi teks. Arsitektur ini juga mengintegrasikan komponen algoritma pencarian sebagai modul terpisah yang dapat dipanggil sesuai kebutuhan, memastikan fleksibilitas dalam pemilihan algoritma dan kemudahan maintenance. Integrasi database MySQL memungkinkan penyimpanan data yang terstruktur dan efisien, sementara sistem file management menangani organisasi dokumen CV dalam direktori yang teratur untuk akses yang cepat dan reliabel.

3.4 Contoh Ilustrasi Kasus

Sebagai ilustrasi penggunaan aplikasi, misalkan seorang HR manager ingin mencari kandidat yang memiliki keahlian "Python", "React", dan "SQL". Pengguna akan memasukkan kata kunci tersebut pada form pencarian, memilih algoritma KMP sebagai metode pencarian, dan mengatur untuk menampilkan 10 kandidat teratas. Sistem akan mengekstrak teks dari seluruh CV dalam database, melakukan pencarian exact matching menggunakan algoritma KMP untuk setiap kata kunci. Jika ditemukan kecocokan sempurna, sistem akan menghitung skor berdasarkan frekuensi kemunculan kata kunci. Jika tidak ditemukan exact match untuk beberapa kata kunci, sistem akan melakukan fuzzy matching menggunakan Levenshtein Distance untuk menemukan kata-kata

yang mirip. Hasil akhir akan menampilkan daftar kandidat yang diurutkan berdasarkan total skor kecocokan, lengkap dengan informasi ringkasan dan opsi untuk melihat CV lengkap.



The screenshot shows a web application window titled "CVSearch". On the left is a sidebar with the "CVMatcher" logo. It contains three sections: "Keywords:" with a text input field containing "React, Express, HTML"; "Search Algorithm:" with a toggle switch between "KMP" (selected) and "BM", and a link for "Aho-Corasick"; and "Top Matches:" with a text input field containing "3". A "Search" button is at the bottom of the sidebar. The main content area on the right is light gray and displays the text "Welcome to CVMatcher!".

Gambar 10. Contoh Input

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma

4.1.1 Implementasi Algoritma KMP

search_algorithm.py: kmp_search()

```
def kmp_search(self, search_start=0):
    b = self.kmp_border_func()
    i = search_start
    j = 0
    while (i < self.text_length):
        if (self.text[i].lower() ==
self.keyword[j].lower()):
            if (j == self.keyword_length-1):
                return i - self.keyword_length + 1
            i += 1
            j += 1
        elif (j > 0):
            j = b[j-1]
        else:
            i += 1
    return -1
```

search_algorithm.py: kmp_border_func()

```
def kmp_border_func(self):
    b = [0] * self.keyword_length
    j = 0
    i = 1

    while (i < self.keyword_length):
        if (self.keyword[j] == self.keyword[i]):
            b[i] = j + 1
            i += 1
            j += 1
        elif (j > 0):
            j = b[j-1]
        else:
            b[i] = 0
            i += 1
    return b
```

4.1.2 Implementasi Algoritma BM

search_algorithm.py: boyer_moore_search()

```
def boyer_moore_search(self, search_start=0):
    last = self.build_last_occur()
    n = self.text_length
    m = self.keyword_length
    i = search_start + m - 1

    if i >= n:
        return -1

    j = m - 1

    while i < n:
        if self.keyword[j].lower() ==
self.text[i].lower():
            if j == 0:
                return i
            else:
                i -= 1
                j -= 1
        else:
            lo = last[self.text[i]] if self.text[i]
in last else -1
            i += m - min(j, 1 + lo)
            j = m - 1

    return -1
```

search_algorithm.py: build_last_occur()

```
def build_last_occur(self):
    last = {}
    for i in range(self.keyword_length):
        last[self.keyword[i]] = i
    return last
```

4.1.3 Implementasi Algoritma Aho-Corasick

aho_corasick.py

```
from collections import deque, defaultdict
from .trie import *
```

```

class AhoCorasickSearch:
    def __init__(self, text, keywords):
        self.text = text.lower()
        self.keywords = [kw.lower() for kw in keywords]
        self.trie = Trie(self.keywords)
        self.root = self.trie.root
        self._augment_nodes(self.root)
        self._build_failure_links()

    def _augment_nodes(self, node):
        """Recursively augment nodes with `fail` and
`outputs`."""
        node.fail = None
        node.outputs = []
        if node.is_end_of_word:
            node.outputs.append(self._get_word_from_node(node))
            for child in node.children.values():
                self._augment_nodes(child)

    def _get_word_from_node(self, node):
        """Backtrack to reconstruct the keyword ending at
this node."""
        chars = []
        while node and node.value:
            chars.append(node.value)
            node = node.parent
        return ''.join(reversed(chars))

    def _build_failure_links(self):
        queue = deque()
        for child in self.root.children.values():
            child.fail = self.root
            queue.append(child)
            child.parent = self.root # manually set
parent for path backtracking

        while queue:
            current_node = queue.popleft()
            for char, child in
current_node.children.items():
                queue.append(child)
                fail_node = current_node.fail
                while fail_node and char not in
fail_node.children:
                    fail_node = fail_node.fail
                child.fail = fail_node.children[char] if
fail_node and char in fail_node.children else self.root
                child.outputs += child.fail.outputs
                child.parent = current_node # set parent
for reconstruction

```



```

def ah_search_indexes(self):
    result = defaultdict(list)

    for keyword in self.keywords:
        result[keyword] = []

    node = self.root

    for i, char in enumerate(self.text):
        while node and char not in node.children:
            node = node.fail
        if not node:
            node = self.root
            continue
        node = node.children[char]
        for word in node.outputs:
            result[word].append(i - len(word) + 1)

    return dict(result)

```

Berikut juga merupakan struktur Trie yang digunakan untuk processing kata

trie.py

```

class TrieNode:
    def __init__(self, value=None):
        self.value = value
        self.children = {}
        self.is_end_of_word = False

class Trie:
    def __init__(self, keyword_array):
        self.root = TrieNode()
        self.create_trie(keyword_array)

    def create_trie(self, keyword_array):
        for word in keyword_array:
            if word:
                current_node = self.root
                for char in word.lower():
                    if char not in current_node.children:
                        new_node = TrieNode(char)
                        new_node.parent = current_node
                        current_node.children[char] =
new_node
                        current_node =
current_node.children[char]
                        current_node.is_end_of_word = True

```

4.1.4 Implementasi Algoritma Levenshtein Distance

search_algorithm.py: levenshtein_distance()

```
def levenshtein_distance(self, a, b):
    sub_cost = 1
    del_cost = 1
    ins_cost = 1
    m, n = len(a), len(b)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            insertion = dp[i][j - 1] + ins_cost
            deletion = dp[i - 1][j] + del_cost
            substitution = dp[i - 1][j - 1] + (0 if
a[i - 1] == b[j - 1] else sub_cost)
            dp[i][j] = min(insertion, deletion,
substitution)

    return dp[m][n]
```

4.1.5 Implementasi Regular Expression untuk Ekstraksi CV

Pada ekstraksi teks dari file PDF secara mentahan, regex digunakan untuk mengurangi newline (\n) yang berulang.

pdf_text.py: extract_raw_from_pdf()

```
def extract_raw_from_pdf(self):
    """Extracts raw text from all pages of a PDF
file, replacing newlines with spaces."""
    doc = fitz.open(self.pdf_path)
    full_text = ""

    for page in doc:
        text = page.get_text("text")
        full_text += text

    cleaned_text = re.sub(r'[\n\r]+', ' ', full_text)
```

```
return cleaned_text.strip()
```

Selain itu, terjadi preprocessing saat mengambil text dari PDF, yakni untuk indent yang melebihi yang minimum, merupakan pertanda text itu adalah bulletpoint di PDF asli. Karena bulletpoint asli tidak bisa dideteksi dari extract pdf, indent yang dimanfaatkan untuk mendapatkan suatu bulletpoint. Text yang dianggap bullet point diberi wrapping `<list></list>` untuk mempermudah processing selanjutnya:

pdf_text.py: indent_text_to_format_text

```
def indent_text_to_format_text(self, ind_text_arr):
    min_x = math.inf
    for x, y in ind_text_arr:
        if (x < min_x):
            min_x = x

    result = ""
    for x, y in ind_text_arr:
        if (x > min_x):
            result += "<list>" + y + "</list>\n"
        else:
            result += y + "\n"

    return result
```

Selanjutnya kebanyakan regex digunakan pada summary.py yang merupakan file untuk mengambil dari pdf secara langsung untuk memperoleh summary. Berikut adalah fungsi yang mengambil text dan memisahkannya menjadi header dan content. Header ditentukan dari JSON header yang berisi judul dan sinonim dari judul-judul tersebut yang disimpan pada data/header_dictionary.json.

pdf_text.py: get_summary()

```
def get_summary(self):
    header_map = {}
    all_headers = []
    for key, variants in
self.read_header_dic().items():
        for variant in variants:
            norm = variant.lower()
            header_map[norm] = key
            all_headers.append(re.escape(norm))

    pattern =
re.compile(rf"(?<=\n) ({'|'.join(all_headers)}) (?=\n)",
```

```

re.IGNORECASE)

    normalized_text = "\n" + self.text.strip() + "\n"
    matches = list(pattern.finditer(normalized_text))

    result = {}
    for i, match in enumerate(matches):
        header_text = match.group(1).strip().lower()
        # header_key = header_map.get(header_text)
        start = match.end()
        end = matches[i + 1].start() if i + 1 <
len(matches) else len(normalized_text)
        content = normalized_text[start:end].strip()
        result[header_text] = content

    return result

```

Setelah itu, dengan memanfaatkan `<list></list>` yang diikuti huruf kecil, dapat disatukan baris bullet point yang lebih dari satu line. Juga ditambah `<bullet></bullet>` untuk konten yang banyak berisi koma dan tidak merupakan `<list>`

pdf_text.py: raw_summary_filter(summary_dic)

```

def raw_summary_filter(self, summary_dic):
    filtered = {}
    for header, content in summary_dic.items():
        normalized = re.sub(r'\n{2,}', '\n',
content.strip()) # get rid of extra new line
        normalized = re.sub(r'</list>\n<list>[a-z]',
' ', normalized.strip()) # combine long list element
        if '<list>' not in normalized and ',' in
normalized: # make bullets
            items = [item.strip() for item in
normalized.strip(", ").split(',') if item.strip()]
            word_counts = [len(item.split()) for item
in items]
            if all(count <= 5 for count in
word_counts):
                normalized =
'\n'.join(f"<bullet>{item}</bullet>" for item in items)
                filtered[header] = normalized

    return filtered

```

Pada filter terakhir, dimanfaatkan `<list>` dan `<bullet>` untuk disederhanakan menjadi bentuk array

pdf_text.py: final_summary_filter(summary_dic)

```
def final_summary_filter(self, summary_dic):
    filtered = {}
    for header, content in summary_dic.items():
        if '<bullet>' not in content:
            filtered_arr =
self.split_non_list_chunks(content)
            filtered[header] = {"type":
TextFormat.List, "content": filtered_arr}
        else:
            bullet_arr =
self.bullet_block_to_array(content)
            filtered[header] = {"type":
TextFormat.Bullet, "content": bullet_arr}
    return filtered

# handling <list>
def split_non_list_chunks(self, content):
    lines = content.strip().split('\n')

    if all(line.strip().startswith("<list>") and
line.strip().endswith("</list>") for line in lines if
line.strip()):
        return [re.sub(r"</?list>", "", line.strip())
for line in lines if line.strip()]

    result = []
    buffer = []

    for line in lines:
        line = line.strip()
        if line.startswith("<list>") and
line.endswith("</list>"):
            if buffer:
                joined = "\n".join(buffer).strip()
                if joined:
                    result.append(joined)
                    buffer = [] # reset after chunk
            else:
                buffer.append(line)

# handling <bullet>
def bullet_block_to_array(self, bullet_block):
    return re.findall(r'<bullet>(.*?)</bullet>',
bullet_block.strip(), re.DOTALL)
```

4.1.6 Implementasi Enkripsi Data

class SimpleEncryption

```
CLASS SimpleEncryption:

    // Constructor
    CONSTRUCTOR():
        SET shift = 13 // ROT13 style shift value
        SET charset =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123
456789 ., '- "

    // Encryption method
    FUNCTION encrypt(message):
        IF message is empty:
            RETURN message

        // Step 1: Convert message to Base64
        message_bytes = CONVERT message TO bytes using
UTF-8 encoding
        base64_encoded = ENCODE message_bytes TO Base64
string

        // Step 2: Apply Caesar cipher to Base64 string
        result = EMPTY array

        FOR each character IN base64_encoded:
            IF character is alphanumeric:
                IF character is digit (0-9):
                    shifted = (character_as_integer +
shift) MOD 10
                    APPEND shifted TO result
                ELSE IF character is uppercase letter
(A-Z):
                    shifted = ((character_ASCII -
'A'_ASCII + shift) MOD 26) + 'A'_ASCII
                    APPEND ASCII_to_character(shifted) TO
result
                ELSE IF character is lowercase letter
(a-z):
                    shifted = ((character_ASCII -
'a'_ASCII + shift) MOD 26) + 'a'_ASCII
                    APPEND ASCII_to_character(shifted) TO
result
            ELSE:
                APPEND character TO result
        ELSE:
            APPEND character TO result // Special
characters unchanged
```

```

RETURN JOIN result as string

// Decryption method
FUNCTION decrypt(encrypted_message):
    IF encrypted_message is empty:
        RETURN encrypted_message

    TRY:
        // Step 1: Reverse Caesar cipher
        result = EMPTY array

        FOR each character IN encrypted_message:
            IF character is alphanumeric:
                IF character is digit (0-9):
                    shifted = (character_as_integer -
shift) MOD 10
                    APPEND shifted TO result
                ELSE IF character is uppercase letter
(A-Z):
                    shifted = ((character_ASCII -
'A'_ASCII - shift) MOD 26) + 'A'_ASCII
                    APPEND
ASCII_to_character(shifted) TO result
                ELSE IF character is lowercase letter
(a-z):
                    shifted = ((character_ASCII -
'a'_ASCII - shift) MOD 26) + 'a'_ASCII
                    APPEND
ASCII_to_character(shifted) TO result
                ELSE:
                    APPEND character TO result
            ELSE:
                APPEND character TO result

        base64_decoded_str = JOIN result as string

        // Step 2: Decode from Base64
        decoded_bytes = DECODE base64_decoded_str
FROM Base64
        original_message = CONVERT decoded_bytes TO
string using UTF-8 encoding

        RETURN original_message

    CATCH any exception:
        PRINT "Decryption error for '" +
encrypted_message + "': " + exception_message
        RETURN encrypted_message // Return original
if decryption fails

```

```
END CLASS
```

****SimpleEncryption**** menggunakan skema enkripsi berlapis dua yang menggabungkan Base64 encoding dan Caesar Cipher dengan shift value 13 (ROT13). Proses enkripsi dimulai dengan mengkonversi pesan asli ke UTF-8 bytes, kemudian di-encode ke Base64 untuk menghasilkan string yang hanya berisi karakter A-Z, a-z, 0-9, dan simbol khusus. Selanjutnya, setiap karakter alfanumerik dalam string Base64 tersebut di-shift menggunakan Caesar Cipher: huruf besar dan kecil di-shift 13 posisi dengan modulo 26, angka di-shift 13 posisi dengan modulo 10, sedangkan karakter khusus (+, /, =) dibiarkan tidak berubah. Proses dekripsi melakukan kebalikannya: pertama reverse Caesar Cipher untuk mengembalikan string Base64 asli, kemudian decode Base64 ke bytes dan konversi kembali ke UTF-8 string. Meskipun tidak cocok untuk data sangat sensitif, skema ini efektif untuk obfuscation ringan dan perlindungan dasar karena menggabungkan dua layer yang membuat data tidak mudah dibaca secara casual, sambil tetap mempertahankan kesederhanaan implementasi dan error handling yang baik.

4.2 Penggunaan Program

4.2.1 Setup Database MySQL

Sebagai inisialisasi awal, hal yang harus diperhatikan adalah kesesuaian tabel yang digunakan dengan berikut ini.

```
CREATE TABLE ApplicantProfile (  
    applicant_id INT AUTO_INCREMENT PRIMARY KEY,  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    date_of_birth DATE,  
    address VARCHAR(255),  
    phone_number VARCHAR(20)  
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;  
  
CREATE TABLE ApplicationDetail (  
    detail_id INT AUTO_INCREMENT PRIMARY KEY,  
    applicant_id INT NOT NULL,  
    application_role VARCHAR(100),  
    cv_path TEXT,  
    FOREIGN KEY (applicant_id) REFERENCES  
ApplicantProfile(applicant_id)  
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```


Untuk melakukan enkripsi dapat dengan perintah berikut.

```
python src/encrypt.py
```

Perintah tersebut akan otomatis membuat seluruh sql terenripsi.

4.2.2 Setup Environment

1. Injeksi SQL seeding

Dari root directory, pertama harus memasukkan file .sql dari seeding dataset, hal tersebut dapat dilakukan dengan command berikut, user yang digunakan harus root pada mysql:

```
mysql -u root -p < data/tubes3_seeding.sql
```

2. Setup File .env

Pada root directory terdapat file yang bernama “.envi”. File tersebut harus diisi dengan password asli dari mysql untuk user “root”, semisal password adalah “sql”, maka isi sebagai berikut:

```
DB_PASSWORD=sql
```

Setelah mengganti passwordnya, ubah nama file dari “.envi” menjadi “.env”

3. Instalasi Library Python

Jalankan command-command berikut secara berurutan untuk instalasi semua library python yang diperlukan untuk menjalankan program. Catatan: untuk pengaktifan venv python mungkin berbeda tergantung sistem, bisa tidak perlu menggunakan “source”.

```
python -m venv venv  
source venv/bin/activate # or "source venv/Scripts/activate" on Windows  
pip install -r requirements.txt
```

4. Download Dataset Kaggle

Terakhir adalah download dataset dari Kaggle untuk menggunakan dataset sesuai yang digunakan untuk pengembangan aplikasi ini

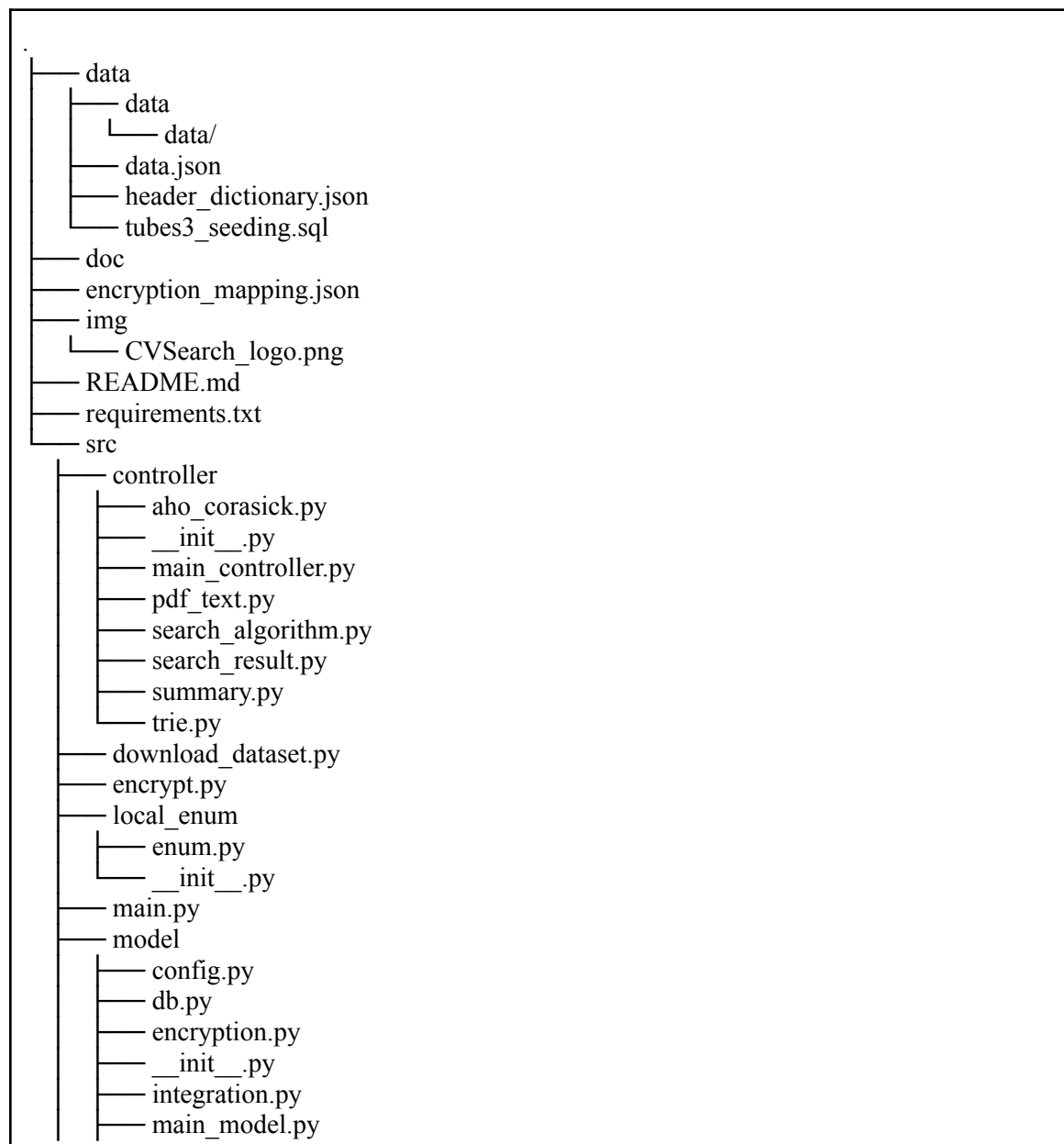
```
python src/download_dataset.py
```

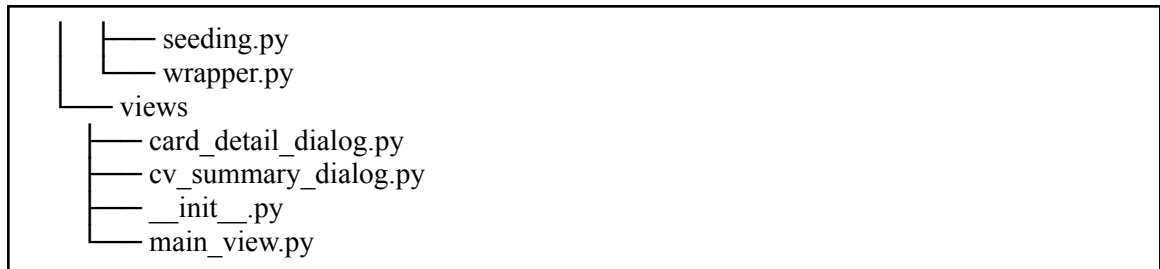
4.2.3 Menjalankan Program

Setelah menjalankan semua langkah setup, cukup menjalankan line berikut untuk menjalani program.

```
python src/main.py
```

4.3 Struktur File





4.4 Hasil Pengujian

4.4.1 Algoritma KMP

Kata Kunci : React

The screenshot displays the CVSearch application interface. On the left, there is a sidebar with the following elements: a logo for 'CVMatcher', a 'Keywords:' input field containing 'react', a 'Search Algorithm:' section with a toggle switch for 'KMP' (selected) and 'BH', and a 'Top Matches:' section with a value of '20'. A 'Search' button is located at the bottom of the sidebar. The main area is titled 'Search Results' and shows '600 Cvs scanned in 3.90s'. It contains six result cards arranged in a 2x3 grid. Each card displays a candidate name and ID, the number of matches, and a list of matched keywords.

Candidate	Matches	Matched keywords
H41k4l 455y4uq1	4 matches	• react: 4 occurrences
HA1K4L ASSY4UQ1	1 match	• react: 1 occurrence
h41K4L Assy4Uq1	1 match	• react: 1 occurrence
RaDen FrAnC1SCo	1 match	• react: 1 occurrence
AR13L H3RFR1SON	1 match	• react: 1 occurrence
AR13L H3RFR1SON	1 match	• react: 1 occurrence

Navigation buttons: Previous, Next

HA1K4L ASSY4UQ1

1 MATCH

Matched Keywords

react

1

1


1 exact occurrence

Close

Summary

View CV

Kata Kunci : Html, Java

 OLMatches

Keywords:
Html, Java

Search Algorithm:
KHP ☐ BH ☒
Also Corrank

Top Matches:
20

Search

Search Results

600 Cvs scanned in 5.31s

Aland MuL1A 10 matches

Matched keywords:

- Html: 4 occurrences
- Java: 6 occurrences

Aland MuL1A 10 matches

Matched keywords:

- Html: 4 occurrences
- Java: 6 occurrences

4L4ND MUL1A 7 matches

Matched keywords:

- Html: 4 occurrences
- Java: 3 occurrences

4l4nd MuL14 6 matches

Matched keywords:

- Html: 2 occurrences
- Java: 4 occurrences

MoH4mM4d NugR4h4 5 matches

Matched keywords:

- Html: 3 occurrences
- Java: 2 occurrences

MoH4mM4d NugR4h4 5 matches

Matched keywords:

- Html: 3 occurrences
- Java: 2 occurrences

Previous Next

Aland MuL1A

10 MATCHES

Matched Keywords

1 **Html**
4 exact occurrences

4

2 **Java**
6 exact occurrences


6

Close

Summary

View CV

Kata Kunci : Data Scientist, Java, Machine Learning, SQL



Keywords:

Data Scientist, Java, Machine Learning, SQL

Search Algorithm:

KHP

☐

BH

Also Consider

Top Matches:

20

Search

Search Results
609 CVs scanned in 11.03s

1khwan
4lh4k1m
11 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 4 occurrences
- Machine Learning: 2 occurrences
- ... and 1 more keyword

4riel Herfris0n
8 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 2 occurrences
- Machine Learning: 2 occurrences
- ... and 1 more keyword

4riel Herfris0n
8 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 2 occurrences
- Machine Learning: 2 occurrences
- ... and 1 more keyword

H41k4l
455y4uq1
12 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 2 occurrences
- Machine Learning: 0 occurrences
- ... and 1 more keyword

H41k4l
455y4uq1
12 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 2 occurrences
- Machine Learning: 0 occurrences
- ... and 1 more keyword

H41k4l
455y4uq1
12 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 2 occurrences
- Machine Learning: 0 occurrences
- ... and 1 more keyword

Previous

Next

36

1khwan 4lh4k1m

11 MATCHES

Matched Keywords

2 **Java**
4 exact occurrences

3 **Machine Learning**
2 exact occurrences

Close

Summary

View CV

Kata Kunci : senior analyst, programming, IT, java, advanced

Keywords:

senior analyst, programming, IT, java, advanced

Search Algorithm:

KHP ☐ BH ☒

Also Consider

Top Matches:

20

Search

Search Results

609 CVs scanned in 10.55s

**RAD3N
FRANC15CO** 62 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 1 occurrence
- IT: 58 occurrences
- ... and 2 more keywords

**RAD3N
FRANC15CO** 62 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 1 occurrence
- IT: 58 occurrences
- ... and 2 more keywords

**RAD3N
FRANC15CO** 62 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 1 occurrence
- IT: 58 occurrences
- ... and 2 more keywords

Aland MuL1A 54 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 5 occurrences
- IT: 42 occurrences
- ... and 2 more keywords

Aland MuL1A 54 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 5 occurrences
- IT: 42 occurrences
- ... and 2 more keywords

**H41k4l
455y4uq1** 29 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 4 occurrences
- IT: 22 occurrences
- ... and 2 more keywords

Previous

Next

RAD3N FRANC15CO

62 MATCHES

Matched Keywords

2

programming
1 exact occurrence

1

3

IT
58 exact occurrences

58

4

java
1 exact occurrence

1

Close


Summary

View CV

4.4.2 Algoritma BM

Kata Kunci : React

CVSearch



Keywords:

Search Algorithm:

KMP

☐

BH

Aho Corasick

☒

Top Matches:

Search

Search Results
600 CVs scanned in 4.11s

H41k4l 455y4uq1
3 matches

Matched keywords:
• react: 3 occurrences

HA1K4L ASSY4UQ1
1 match

Matched keywords:
• react: 1 occurrence

h41K4L Assy4Uq1
1 match

Matched keywords:
• react: 1 occurrence

RaDen FrAnC1SCo
1 match

Matched keywords:
• react: 1 occurrence

AR13L H3RFR1SON
1 match

Matched keywords:
• react: 1 occurrence

AR13L H3RFR1SON
1 match

Matched keywords:
• react: 1 occurrence

Previous

Next

HA1K4L ASSY4UQ1

1 MATCH

Matched Keywords

react

1

1

1 exact occurrence

Close

Summary

View CV

Kata Kunci : Html, Java

OLMatches

Keywords:
Html, Java

Search Algorithm:
KMP ☒ BH ☐
Also Consider

Top Matches:
20

Search

Search Results
600 CVs scanned in 4.75s

Aland MuL1A
9 matches

Matched keywords:

- Html: 3 occurrences
- Java: 6 occurrences

Aland MuL1A
9 matches

Matched keywords:

- Html: 3 occurrences
- Java: 6 occurrences

4l4nd MuL14
6 matches

Matched keywords:

- Html: 2 occurrences
- Java: 4 occurrences

**MoH4mM4d
NugR4h4**
5 matches

Matched keywords:

- Html: 3 occurrences
- Java: 2 occurrences

**MoH4mM4d
NugR4h4**
5 matches

Matched keywords:

- Html: 3 occurrences
- Java: 2 occurrences

**MoH4mM4d
NugR4h4**
5 matches

Matched keywords:

- Html: 3 occurrences
- Java: 2 occurrences

Previous

Next

Aland MuL1A

9 MATCHES

Matched Keywords

1

Html

3 exact occurrences

3

2

Java

6 exact occurrences

6

Close

Summary

View CV

Kata Kunci : Data Scientist, Java, Machine Learning, SQL

40

AlMatcha

Keywords:
Data Scientist, Java, Machine Learning, SQL

Search Algorithm:
KIP ☐ BPI ☐
Also Consider

Top Matches:
20

Search

Search Results
600 CVs scanned in 10.81s

**1khwan
4lh4k1m**
11 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 4 occurrences
- Machine Learning: 2 occurrences
- ... and 1 more keyword

4riel Herfris0n
8 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 2 occurrences
- Machine Learning: 2 occurrences
- ... and 1 more keyword

4riel Herfris0n
8 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 2 occurrences
- Machine Learning: 2 occurrences
- ... and 1 more keyword

**H41k4l
455y4uq1**
12 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 2 occurrences
- Machine Learning: 0 occurrences
- ... and 1 more keyword

**H41k4l
455y4uq1**
12 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 2 occurrences
- Machine Learning: 0 occurrences
- ... and 1 more keyword

**H41k4l
455y4uq1**
12 matches

Matched keywords:

- Data Scientist: 0 occurrences
- Java: 2 occurrences
- Machine Learning: 0 occurrences
- ... and 1 more keyword

Previous

Next

1khwan 4lh4k1m
11 MATCHES

Matched Keywords

1

Data Scientist
0 occurrences

0

2

Java
4 exact occurrences

4

3

Machine Learning
2 exact occurrences

2

Close

Summary

View CV

Kata Kunci : senior analyst, programming, IT, java, advanced

41

Keywords:

senior analyst, programming, IT, java, advanced

Search Algorithm:

KHP ☒ BH ☐

Aho-Corasick

Top Matches:

20

Search

Search Results

600 CVs scanned in 10.46s

Aland MuL1A 34 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 5 occurrences
- IT: 24 occurrences
- ... and 2 more keywords

Aland MuL1A 34 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 5 occurrences
- IT: 24 occurrences
- ... and 2 more keywords

RAD3N FRANC15CO 27 matches

Matched keywords:

- senior analyst: 0 occurrences
- programmina: 1 occurrence
- IT: 23 occurrences
- ... and 2 more keywords

RAD3N FRANC15CO 27 matches

Matched keywords:

- senior analyst: 0 occurrences
- programmina: 1 occurrence
- IT: 23 occurrences
- ... and 2 more keywords

RAD3N FRANC15CO 27 matches

Matched keywords:

- senior analyst: 0 occurrences
- programmina: 1 occurrence
- IT: 23 occurrences
- ... and 2 more keywords

H41k4l 455y4uq1 18 matches

Matched keywords:

- senior analyst: 0 occurrences
- programmina: 4 occurrences
- IT: 12 occurrences
- ... and 2 more keywords

Previous Next

Aland MuL1A

34 MATCHES

Matched Keywords

3

11

24 exact occurrences

24

4

java

4 exact occurrences

4

5

advanced

1 exact occurrence

1

Close

Summary

View CV

4.4.3 Algoritma Aho-Corasick

Kata Kunci : React

Keywords:

react

Search Algorithm:

KXP

BH

Aho-Corasick

Top Matches:

20

Search

Search Results

600 CVs scanned in 4.37s

H41k4l 455y4uq1

4 matches

Matched keywords:

• react: 4 occurrences

HA1K4L ASSY4UQ1

1 match

Matched keywords:

• react: 1 occurrence

h41K4L Assy4Uq1

1 match

Matched keywords:

• react: 1 occurrence

RaDen FrAnC1SCo

1 match

Matched keywords:

• react: 1 occurrence

AR13L H3RFR1SON

1 match

Matched keywords:

• react: 1 occurrence

AR13L H3RFR1SON

1 match

Matched keywords:

• react: 1 occurrence

Previous

Next

HA1K4L ASSY4UQ1

1 MATCH

Matched Keywords

react

1

1 exact occurrence

Close

Summary

View CV

Kata Kunci : Html, Java

43

Keywords:

html, java

Search Algorithm:

KHP

BH

Also Consider

Top Matches:

20

Search

Search Results

600 CVs scanned in 5.09s

Aland MuL1A

10 matches

Matched keywords:

- html: 4 occurrences
- java: 6 occurrences

Aland MuL1A

10 matches

Matched keywords:

- html: 4 occurrences
- java: 6 occurrences

4L4ND MUL1A

7 matches

Matched keywords:

- html: 4 occurrences
- java: 3 occurrences

4l4nd MuL14

6 matches

Matched keywords:

- html: 2 occurrences
- java: 4 occurrences

**MoH4mM4d
NugR4h4**

5 matches

Matched keywords:

- html: 3 occurrences
- java: 2 occurrences

**MoH4mM4d
NugR4h4**

5 matches

Matched keywords:

- html: 3 occurrences
- java: 2 occurrences

Previous

Next

Aland MuL1A

10 MATCHES

Matched Keywords

1

html

4 exact occurrences

4

2

java

6 exact occurrences

6

Close

Summary

View CV

Kata Kunci : Data Scientist, Java, Machine Learning, SQL

CVSearch

Keywords:

Data Scientist, Java, Machine Learning, SQL

Search Algorithm:

KHP

BH

Also Consider

Top Matches:

20

Search

Search Results

600 CVs scanned in 10.47s

1khwan 4lh4k1m

11 matches

Matched keywords:

- data scientist: 0 occurrences
- java: 4 occurrences
- machine learning: 2 occurrences
- ... and 1 more keyword

4riel HerfrisOn

8 matches

Matched keywords:

- data scientist: 0 occurrences
- java: 2 occurrences
- machine learning: 2 occurrences
- ... and 1 more keyword

4riel HerfrisOn

8 matches

Matched keywords:

- data scientist: 0 occurrences
- java: 2 occurrences
- machine learning: 2 occurrences
- ... and 1 more keyword

H41k4l 455y4uq1

12 matches

Matched keywords:

- data scientist: 0 occurrences
- java: 2 occurrences
- machine learning: 0 occurrences
- ... and 1 more keyword

H41k4l 455y4uq1

12 matches

Matched keywords:

- data scientist: 0 occurrences
- java: 2 occurrences
- machine learning: 0 occurrences
- ... and 1 more keyword

H41k4l 455y4uq1

12 matches

Matched keywords:

- data scientist: 0 occurrences
- java: 2 occurrences
- machine learning: 0 occurrences
- ... and 1 more keyword

Previous

Next

1khwan 4lh4k1m

11 MATCHES

Matched Keywords

2

java

4 exact occurrences

4

3

machine learning

2 exact occurrences

2

4

sql

5 exact occurrences

5

Close

Summary

View CV

Kata Kunci : senior analyst, programming, IT, java, advanced

45

Keywords:

senior analyst, programming, IT, java, advanced

Search Algorithm:

KHP ☒ BH ☐

Aho Corasick

Top Matches:

20

Search

Search Results

600 CVs scanned in 9.95s

RAD3N FRANC15CO

62 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 1 occurrence
- it: 58 occurrences
- ... and 2 more keywords

RAD3N FRANC15CO

62 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 1 occurrence
- it: 58 occurrences
- ... and 2 more keywords

RAD3N FRANC15CO

62 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 1 occurrence
- it: 58 occurrences
- ... and 2 more keywords

Aland MuL1A

54 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 5 occurrences
- it: 42 occurrences
- ... and 2 more keywords

Aland MuL1A

54 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 5 occurrences
- it: 42 occurrences
- ... and 2 more keywords

H41k4l 455y4uq1

29 matches

Matched keywords:

- senior analyst: 0 occurrences
- programming: 4 occurrences
- it: 22 occurrences
- ... and 2 more keywords

Previous Next

RAD3N FRANC15CO

62 MATCHES

Matched Keywords

2

programming

1 exact occurrence

1

3

it

58 exact occurrences

58

4

java

1 exact occurrence

1


Close

Summary

View CV

4.4.4 Fuzzy Matching

Kata Kunci : reavt



Keywords:

Search Algorithm:

☒ KHP
 ☐ BH

Also Consider

Top Matches:

Search

Search Results
600 CVs scanned in 4.03s

FARH4N N4f15 22 matches

Matched keywords:

- reavt: 22 occurrences

FARH4N N4f15 22 matches

Matched keywords:

- reavt: 22 occurrences

Haikal ASSYAUQI 13 matches

Matched keywords:

- reavt: 13 occurrences

Haikal ASSYAUQI 13 matches

Matched keywords:

- reavt: 13 occurrences

RaDen FrAnC1SCo 12 matches

Matched keywords:

- reavt: 12 occurrences


RaDen FrAnC1SCo 12 matches

Matched keywords:

- reavt: 12 occurrences

Previous Next

Kata Kunci : html, jaca



Keywords:

Search Algorithm:

☒ KHP
 ☐ BH

Also Consider

Top Matches:

Search

Search Results
600 CVs scanned in 4.83s

RAD3N FRANC1SCO 36 matches

Matched keywords:

- html: 3 occurrences
- jaca: 33 occurrences

RAD3N FRANC1SCO 36 matches

Matched keywords:

- html: 3 occurrences
- jaca: 33 occurrences

RAD3N FRANC1SCO 36 matches

Matched keywords:

- html: 3 occurrences
- jaca: 33 occurrences

R4d3N Fr4nc1Sco 34 matches

Matched keywords:

- html: 2 occurrences
- jaca: 32 occurrences

R4d3N Fr4nc1Sco 34 matches

Matched keywords:

- html: 2 occurrences
- jaca: 32 occurrences

r4d3n Francisco 24 matches

Matched keywords:

- html: 19 occurrences
- jaca: 5 occurrences

Previous Next

Kata Kunci : Data Engineerig, Jawa, Teknik Machine, PQL

47

Keywords:

Data Engineering, Java, Teknik Machine, PGL

Search Algorithm:

KHP

BH

Alko-Corastick

Top Matches:

20

Search

Search Results

600 CVs scanned in 10.74s

MoH4mM4d
NugR4h4

15 matches

Matched keywords:

- data engineer: 0 occurrences
- java: 2 occurrences
- teknik machine: 0 occurrences
- ... and 1 more keyword

MoH4mM4d
NugR4h4

15 matches

Matched keywords:

- data engineer: 0 occurrences
- java: 2 occurrences
- teknik machine: 0 occurrences
- ... and 1 more keyword

MoH4mM4d
NugR4h4

15 matches

Matched keywords:

- data engineer: 0 occurrences
- java: 2 occurrences
- teknik machine: 0 occurrences
- ... and 1 more keyword

4l4nd MuL14

12 matches

Matched keywords:

- data engineer: 0 occurrences
- java: 2 occurrences
- teknik machine: 0 occurrences
- ... and 1 more keyword

Ikhw4n
ALH4KIM

110 matches

Matched keywords:

- data engineer: 0 occurrences
- java: 10 occurrences
- teknik machine: 0 occurrences
- ... and 1 more keyword

Al4nd Muli4

91 matches

Matched keywords:

- data engineer: 0 occurrences
- java: 24 occurrences
- teknik machine: 0 occurrences
- ... and 1 more keyword

Previous

Next

Kata Kunci : Rusdi

Keywords:

Rusdi

Search Algorithm:

KHP

BH

Alko-Corastick

Top Matches:

20

Search

Search Results

600 CVs scanned in 3.99s

Ari3L HerfR150n

7 matches

Matched keywords:

- rusdi: 7 occurrences

m0h4mm4d
nu9r4h4

4 matches

Matched keywords:

- rusdi: 4 occurrences

RAD3N
FRANC15CO

1 match

Matched keywords:

- rusdi: 1 occurrence

AR13L
H3RFR1S0N

1 match

Matched keywords:

- rusdi: 1 occurrence

AR13L
H3RFR1S0N

1 match

Matched keywords:

- rusdi: 1 occurrence

MoH4mM4d
NugR4h4

1 match

Matched keywords:

- rusdi: 1 occurrence

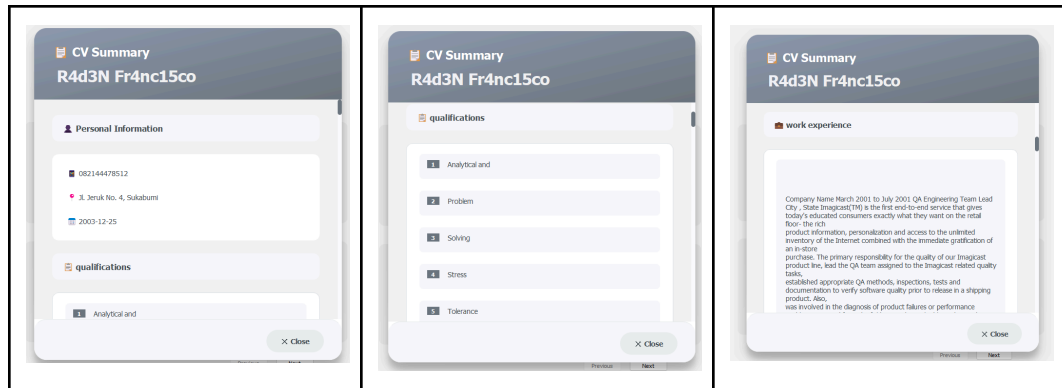
Previous

Next

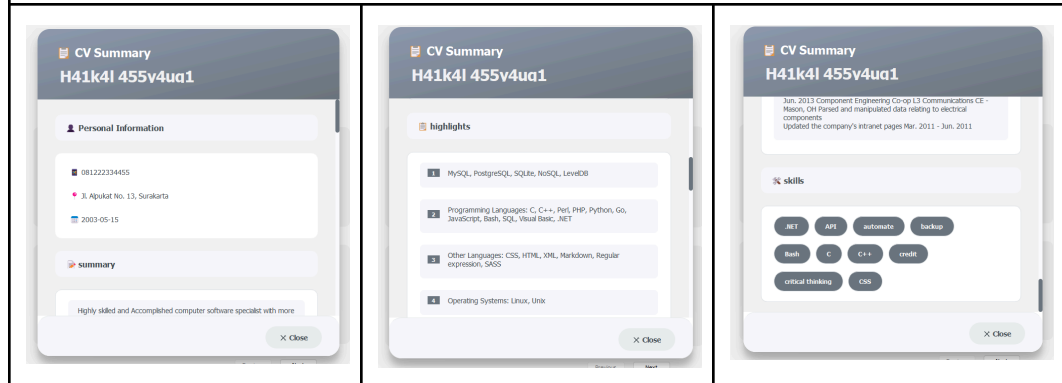
4.4.5 Menampilkan *summary CV applicant*.

Summary dengan pencarian kata kunci : HTML, React

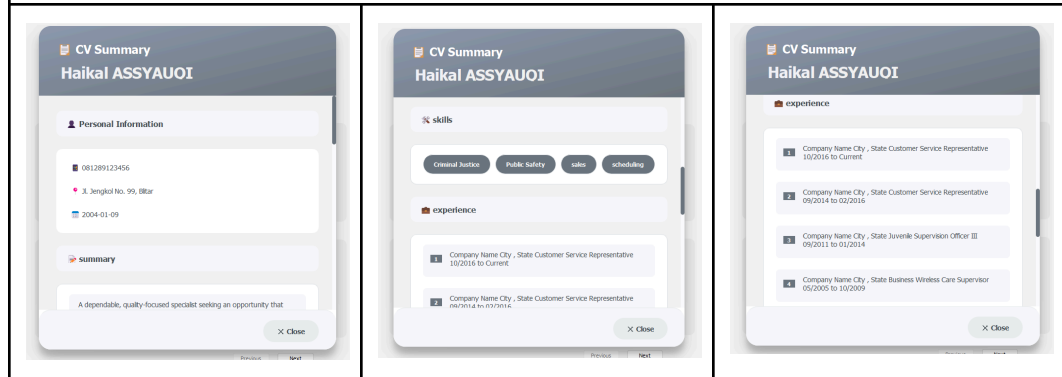
48



Summary dengan pencarian kata kunci : Java, OOP

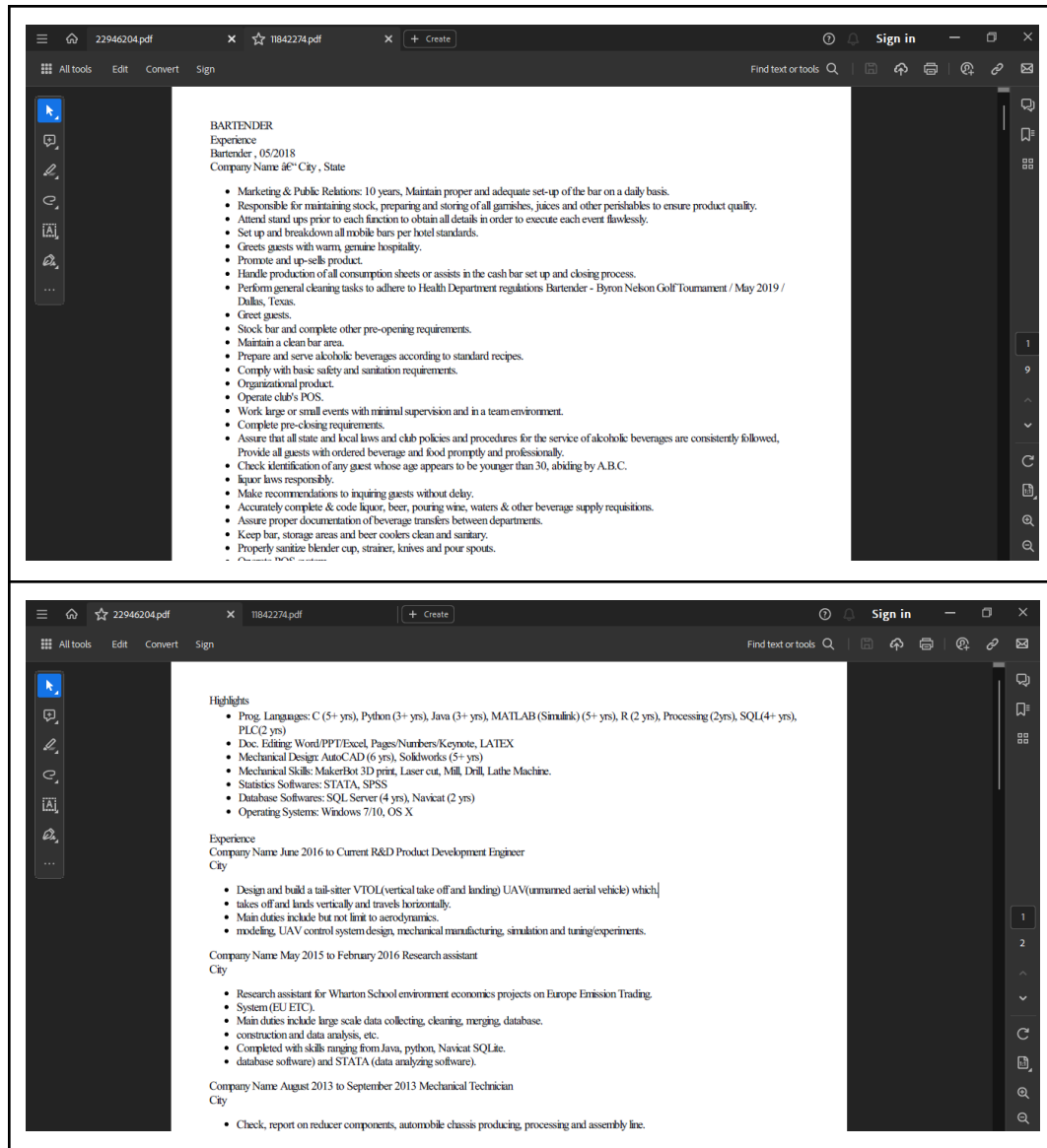


Summary dengan pencarian kata kunci : Cyber Security, Cryptography



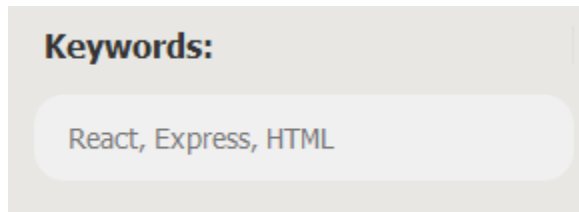
4.4.6 Menampilkan *CV applicant* secara keseluruhan.

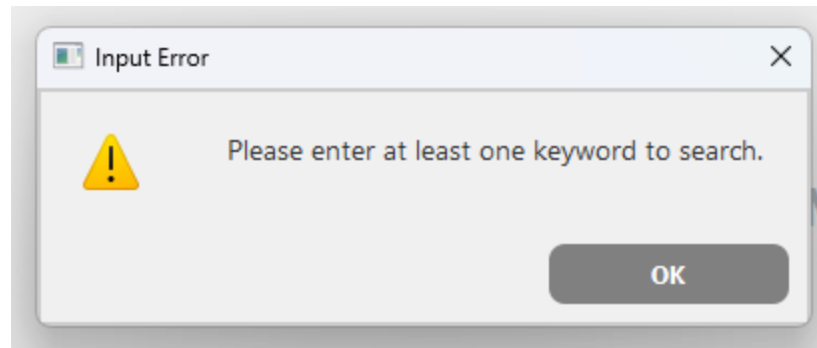
Tampilan CV beberapa applicant



4.4.7 Error Handling

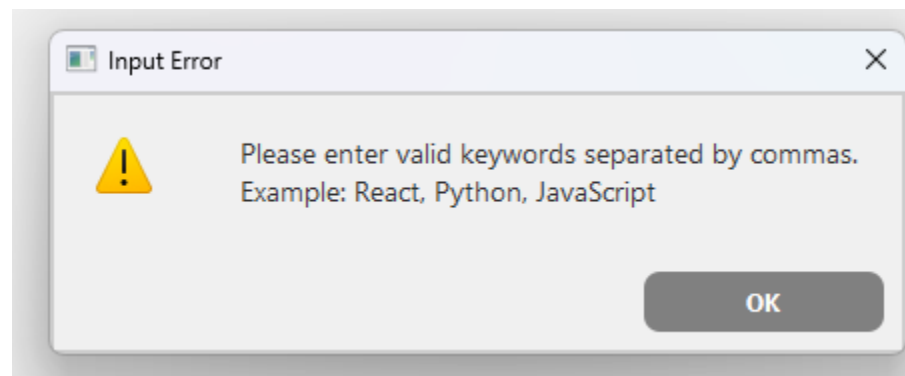
Error message ketika menginput keyword kosong





Error message ketika menginput multiple keyword yang tidak valid

Keywords:



4.4.8 Hasil Enkripsi

```
MariaDB [ats_system]> select * from applicantprofile limit 10;
```

applicant_id	first_name	last_name	date_of_birth	address	phone_number
1	GJ2bAT4gATD=	GaH8pwEbAN==	2003-06-14	FzjhhVRgyozShM5RtGz1hVQRlyPOXLJgupaEu	ZQtKZwZ3AGL6BQxk
2	GH2VAR4AARD=	GyIUHmEVAN==	2004-03-22	FzjhhVR4yoTS3nFOBoL7tAQHfVRWuozE4ozp=	ZQtLZGVmAQH5AmTk
3	GG0bLJ4gLJD=	GaIapwEbAN==	2003-11-05	FzjhhVRAyoJS1LFOBoL7tAljtH6ILLJWurJR=	ZQtKZmD4Amp7BGRl
4	GJ2bLJ4gLJD=	GyIUHmEVAN==	2004-01-17	FzjhhVSAun6ILLFOBoL7tAPjtH5IgLKWuozp=	ZQtLZGZ3AGL6BQxL
5	oG0bAT4gATD=	oaH8pwEbAN==	2003-09-12	FzjhhVR4uq5SLVR8iYvN8YPOMo5q8LJgupaEu	ZQtKZwVmAQH5AmT8
6	GJ2VAT4AATD=	GaIaHwEbAN==	2003-12-30	FzjhhVRShM5qLMJftGz1hVQR7YPOAMJEuot=	ZQtLZGDmZwH5AmT8
7	GG0VAR4AARD=	GaIUpwEVLd==	2003-05-26	FzjhhVRE4n6HtGz1hVQZmYPOALJgupa6Aupt=	ZQtKZwx7AmL4AQZl
8	GJ2bLJ4gLJD=	GyIUHxSVDD==	2004-02-11	FzjhhVRSjMJjtGz1hVQVfVS0uoTIgLzShMj=	ZQtKZmD4Amp8ZGVm
9	oG0VAT4gLJD=	oaIaBJSbAN==	2004-04-19	FzjhhVRcuoJW4VR8iYvNLAfjtGJSfLJ8a	ZQtLZGx7AmL4AQZl
10	GG0bAT4AARD=	GyIUHxSVDD==	2003-07-07	FzjhhVS0cp5ShMLOBoL7tAmLfvREyao0up5SL	ZQtKZmx8BGt7BQp6

10 rows in set (0.000 sec)

4.5 Analisis Hasil Pengujian

Berdasarkan pengujian yang dilakukan, sistem CV Matcher yang dibangun sudah berjalan dengan baik dan konsisten memberikan hasil yang sesuai harapan, baik dari segi waktu eksekusi maupun keakuratan pencarian. Secara teori, algoritma Aho-Corasick memang paling unggul untuk pencarian banyak kata kunci secara bersamaan, karena dirancang khusus untuk multi-pattern matching dan dapat memproses seluruh kata kunci dalam satu kali pemindaian teks. Sementara itu, algoritma KMP dan Boyer-Moore lebih optimal digunakan untuk pencarian satu pola saja, dan akan kalah efisien jika pola yang dicari semakin banyak karena harus dijalankan berulang kali. Namun, dari hasil eksperimen nyata, ketiga algoritma ini memberikan waktu eksekusi yang hampir serupa, misalnya pada salah satu kasus, KMP hanya memerlukan 3,95 ms, Boyer-Moore 4,11 ms, dan Aho-Corasick 4,37 ms.

Pada pengujian dengan fuzzy matching menggunakan Levenshtein Distance, hasil yang diperoleh juga sangat baik, terutama untuk kasus-kasus typo ringan atau perbedaan minor pada kata kunci, sehingga sistem tetap dapat menampilkan hasil yang relevan. Namun, pada beberapa kasus tertentu hasil fuzzy match bisa kosong jika threshold kemiripan yang dipakai terlalu ketat. Ini adalah trade-off yang harus dipertimbangkan dalam penentuan threshold optimal. Secara umum, KMP dan Boyer-Moore menunjukkan hasil waktu yang sangat mirip, dengan keunggulan KMP pada pola pendek atau sangat berulang, sedangkan Boyer-Moore baru terasa lebih cepat pada pola panjang dan variasi alfabet yang lebar. Pada pola satu huruf, semua algoritma memang berjalan lebih lama karena harus melakukan iterasi secara menyeluruh pada teks.

Secara keseluruhan, aplikasi yang dikembangkan telah mampu memenuhi kebutuhan pencarian dan penyaringan CV secara otomatis dan efisien. Penggunaan kombinasi algoritma KMP, Boyer-Moore, dan Aho-Corasick, serta penambahan fitur fuzzy matching, terbukti efektif untuk menyaring ribuan CV berdasarkan kata kunci, sekaligus tetap adaptif terhadap kesalahan input manusia. Hasil ini menegaskan pentingnya pengujian langsung di dunia nyata untuk menilai performa algoritma secara menyeluruh, dan membuktikan bahwa pendekatan pattern matching masih sangat relevan dan aplikatif untuk kebutuhan rekrutmen digital masa kini.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pengembangan aplikasi CV Matching ini membuktikan bahwa penerapan algoritma string matching memiliki potensi besar dalam mendukung proses seleksi kandidat secara efisien dan akurat. Dengan mengintegrasikan teknologi ini, proses pencocokan kebutuhan pekerjaan dengan keahlian kandidat menjadi lebih cepat, terstruktur, dan minim kesalahan manusia. Aplikasi ini telah menunjukkan kemampuan untuk membantu perusahaan dalam mempercepat proses rekrutmen sekaligus memberikan pengalaman pengguna yang lebih baik.

5.2 Saran

Untuk pengembangan lebih lanjut, disarankan untuk meningkatkan akurasi algoritma dengan mempertimbangkan elemen lain seperti analisis semantik dan penggunaan teknologi machine learning. Selain itu, aplikasi dapat dikembangkan lebih jauh dengan menambahkan fitur seperti analisis kompatibilitas budaya kerja, rekomendasi pelatihan untuk meningkatkan keterampilan kandidat, serta laporan statistik yang lebih mendalam untuk perekrut. Penyesuaian terhadap kebutuhan industri yang spesifik juga dapat meningkatkan nilai aplikasi ini.

5.3 Refleksi

Pengembangan aplikasi CV Matching ini menjadi pengalaman yang sangat berharga bagi kami dalam memahami implementasi algoritma string matching di dunia nyata. Selain belajar aspek teknis seperti pengembangan algoritma dan integrasi teknologi, kami juga mendapatkan wawasan tentang pentingnya pengelolaan proyek, kolaborasi tim, dan komunikasi yang efektif. Proses ini mengajarkan kami untuk bekerja dengan berbagai perspektif, saling mendukung, dan menyelesaikan tantangan bersama. Pengalaman ini tidak hanya memperkaya kompetensi teknis kami tetapi juga membentuk keterampilan interpersonal yang penting untuk dunia kerja di masa depan.

LAMPIRAN

Tabel :

No	Poin	Ya	Tidak
1.	Aplikasi dapat dijalankan.	V	
2.	Aplikasi menggunakan basis data berbasis SQL dan berjalan dengan lancar.	V	
3.	Aplikasi dapat mengekstrak informasi penting menggunakan Regular Expression (Regex).	V	
4.	Algoritma <i>Knuth-Morris-Pratt (KMP)</i> dan <i>Boyer-Moore (BM)</i> dapat menemukan kata kunci dengan benar.	V	
5.	Algoritma Levenshtein Distance dapat mengukur kemiripan kata kunci dengan benar.	V	
6.	Aplikasi dapat menampilkan <i>summary CV applicant</i> .	V	
7.	Aplikasi dapat menampilkan <i>CV applicant</i> secara keseluruhan.	V	
8.	Membuat laporan sesuai dengan spesifikasi.	V	
9.	Membuat bonus enkripsi data profil <i>applicant</i> .	V	
10.	Membuat bonus algoritma Aho-Corasick.	V	
11.	Membuat bonus video dan diunggah pada Youtube.	V	

Link Github : https://github.com/Staryo40/Tubes3_Itulah_4_trio

Link Youtube : <https://youtu.be/94xGvIiqHzI>

DAFTAR PUSTAKA

“Aho-Corasick Algorithm for Pattern Searching.” *www.geeksforgeeks.org*,

<https://www.geeksforgeeks.org/dsa/aho-corasick-algorithm-pattern-searching/>. Accessed 15 06 2025.

Munir, Rinaldi. “Pencocokan String (String/Pattern Matching).”

<https://informatika.stei.itb.ac.id/~rinaldi.munir/>,

[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)

f. Accessed 15 06 2025.