

LAPORAN TUGAS BESAR 3

IF2211 STRATEGI ALGORITMA

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



Disusun oleh:

Kelompok 11 - Destroyed

Nama	NIM
Nicholas Andhika Lucas	13523014
Aria Judhistira	13523112
Ignacio Kevin Alberiann	15223090

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

2025

DAFTAR ISI

DAFTAR ISI.....	2
DAFTAR GAMBAR.....	3
DAFTAR TABEL.....	4
BAB 1: Deskripsi Tugas.....	5
BAB 2: Landasan Teori.....	7
2.1 Dasar Teori.....	7
2.2 Program Aplikasi.....	11
BAB 3: Analisis Pemecahan Masalah.....	12
1. Langkah-langkah Pemecahan Masalah.....	12
2. Proses Pemetaan Masalah.....	12
3. Fitur Fungsional dan Arsitektur Aplikasi.....	16
BAB 4: Implementasi dan Pengujian.....	17
4.1. Spesifikasi Teknis Program.....	17
4.2. Tata Cara Penggunaan Program.....	23
4.3. Hasil Pengujian.....	23
4.4. Analisis Hasil Pengujian.....	23
BAB 5: Kesimpulan, Saran, dan Refleksi.....	24
5.1 Kesimpulan.....	24
5.2 Saran.....	24
5.3 Refleksi.....	24
Lampiran.....	25
1. Tautan Repository.....	25
2. Tautan Video.....	25
3. Tabel Checklist.....	25
Daftar Pustaka.....	27

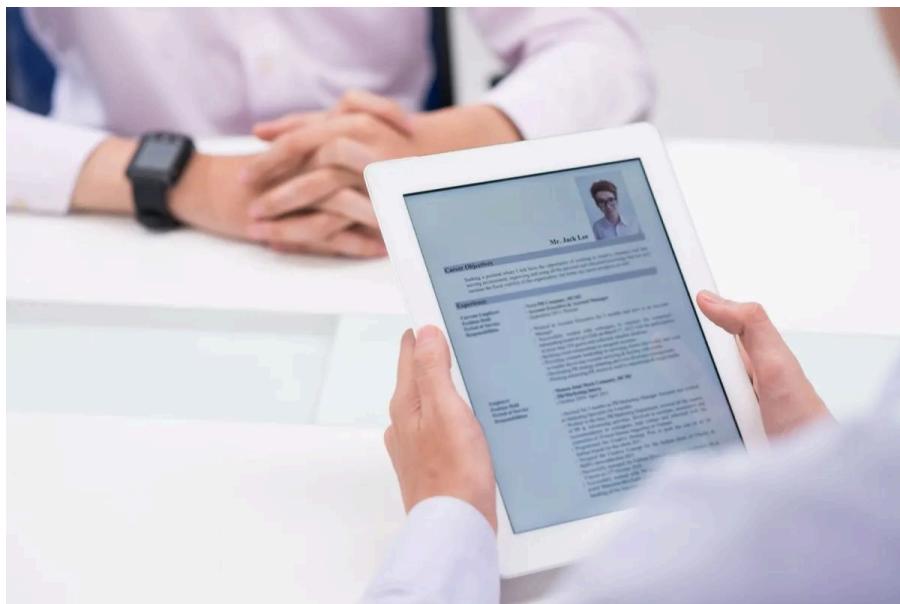
DAFTAR GAMBAR

Gambar 1.1 CV ATS dalam Dunia Kerja.....	5
Gambar 2.1 Ilustrasi algoritma Knuth-Morris-Pratt.....	7
Gambar 2.2 Ilustrasi algoritma Boyer-Moore.....	8
Gambar 2.3 Ilustrasi data trie pada algoritma Aho-Corasick.....	9
Gambar 2.4 Ilustrasi failure link pada node algoritma Aho-Corasick.....	10
Gambar 2.5 Ilustrasi Levenshtein Distance.....	11
Gambar 2.6 Ilustrasi Pohon Rekursi untuk Levenshtein Distance.....	11

DAFTAR TABEL

Tabel 1. Tabel Checklist.....	33
-------------------------------	----

BAB 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.

Pattern matching adalah teknik untuk menemukan dan mencocokkan pola tertentu dalam teks. Dalam konteks ini, algoritma Boyer-Moore dan Knuth-Morris-Pratt (KMP) sering digunakan karena keduanya menawarkan efisiensi tinggi untuk pencarian teks di

dokumen besar. Algoritma ini memungkinkan sistem ATS untuk mengidentifikasi informasi penting dari CV pelamar dengan kecepatan dan akurasi yang optimal.

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

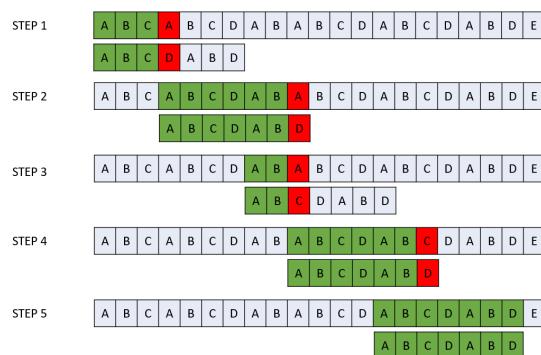
BAB 2: Landasan Teori

2.1 Dasar Teori

Algoritma *pattern matching* atau pencocokan pola merupakan algoritma untuk menemukan sebuah pola dalam teks yang panjang. Dalam kasus pencocokan *string*, pola yang dimaksud merupakan sepotong teks pendek. Pada tugas besar ini, diimplementasikan tiga jenis algoritma pencocokan *string*, yaitu algoritma Knuth-Morris-Pratt (KMP), algoritma Boyer-Moore (BM), dan algoritma Aho-Corasick.

2.1.1 Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) merupakan suatu jenis algoritma pencocokan *string*. Algoritma ini dipublikasikan oleh James H. Morris, Donald Knuth, dan Vaughan Pratt pada tahun 1977. Prinsip utama dari kerja algoritma ini adalah fungsi pinggirannya untuk mencari pola di dalam teks yang ingin dicocokan yang dapat mempercepat proses pencocokan *string*. Pola yang dicari oleh fungsi pinggiran berupa LPS (*Longest Proper Prefix which is also a Suffix*) yang disimpan dalam sebuah tabel.



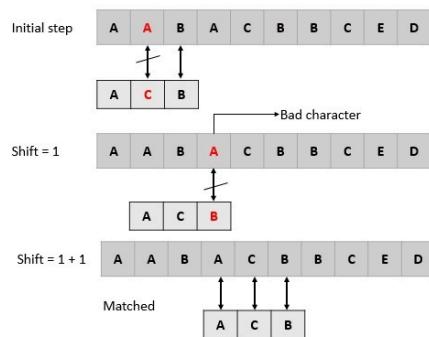
Gambar 2.1 Ilustrasi algoritma Knuth-Morris-Pratt

Pencocokan teks pada algoritma ini dilakukan dari kiri ke kanan. Ketika terjadi ketidakcocokan antara karakter pada pola dengan karakter pada teks, algoritma ini memanfaatkan tabel LPS yang sebelumnya dibuat untuk mengetahui seberapa jauh pola dapat digeser ke kanan tanpa melewatkannya. Nilai dari

tabel LPS memberitahu posisi karakter berikutnya pada pola yang dibandingkan sehingga pergeseran pola terhadap teks dapat dilakukan secara lebih cepat dan intuitif.

2.1.2 Algoritma Boyer-Moore

Algoritma Boyer-Moore (BM) merupakan suatu jenis pencocokan *string* yang dikemukakan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Prinsip kerja algoritma ini memiliki pendekatan yang unik dalam hal pencocokan pola yang, berbeda dengan algoritma KMP, memulai pencocokan dari kanan ke kiri pada pola yang ingin dicocokkan.



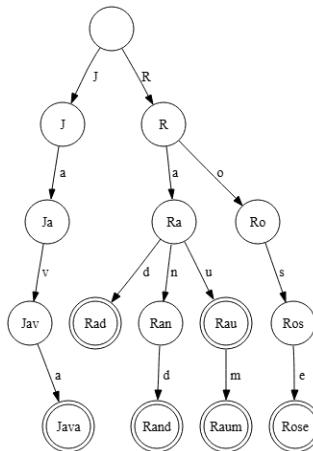
Gambar 2.2 Ilustrasi algoritma Boyer-Moore

Heuristik pergeseran pola pada algoritma ini memanfaatkan fungsi *Last Occurrence* yang mencatat kemunculan terakhir setiap karakter teks yang muncul pada pola. Ketika terjadi ketidakcocokan pada karakter pada teks dengan karakter pada pola, algoritma akan mencari kemunculan terakhir karakter tersebut pada pola dan menggeser kepadanya. Jika pola tidak memiliki karakter tersebut, pola akan digeser melewati karakter tersebut sepenuhnya, dan perbandingan karakter dimulai kembali. Heuristik unik ini membuat algoritma Boyer-Moore bekerja lebih baik pada alfabet yang luas.

2.1.3 Algoritma Aho-Corasick

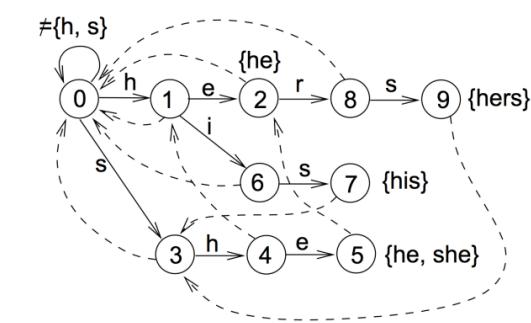
Algoritma Aho-Corasick adalah salah satu algoritma yang dapat digunakan untuk pencocokan *multiple string* yang dikembangkan oleh Alfred V. Aho dan Margaret J. Corasick pada tahun 1975. Algoritma ini berbeda dengan algoritma KMP dan Boyer-Moore yang mencari satu *pattern/string* tertentu dalam sebuah teks,

melainkan algoritma ini diprogram untuk secara efisien mencari beberapa pola (kamus) sekaligus dalam sebuah teks. Prinsip kerja utama dari algoritma ini adalah penggunaan struktur data trie (prefix tree).



Gambar 2.3 Ilustrasi data trie pada algoritma Aho-Corasick

Algoritma ini dimulai dengan pembangunan trie dengan memasukkan semua pola yang ingin dicari ke dalam trie sebagai suatu kamus. Setiap node/simpul dari data trie merepresentasikan prefiks satu atau lebih pola atau akhir dari pola. Dalam algoritma ini pula digunakan sebuah fungsi *failure link* atau *suffix link* yang menjadi bagian paling krusial di mana ketika karakter yang discan tidak cocok dengan jalur dalam trie, algoritma akan menuju node lain yang merupakan prefiks terpanjang dari string yang telah dicocokkan sejauh ini dan merupakan prefiks salah satu pola lain dalam kamus, di mana prefiks tersebut juga merupakan sufiks dari string tersebut. Hal ini ditujukan supaya tidak terjadi pemindaian ulang teks dari awal saat terjadi ketidakcocokan atau *failure* sehingga efisien untuk menemukan semua kecocokan. Dalam proses pencocokan atau *matching*, algoritma akan berpindah dari node ke node mengikuti karakter yang sedang dipindai, dan mengikuti *failure link* apabila terjadi *mismatch* untuk mengidentifikasi pola-pola lain yang mungkin merupakan sufiks dari kecocokan yang baru ditemukan.

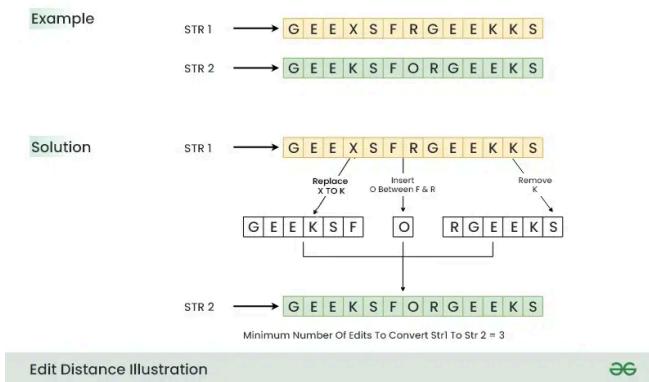


Gambar 2.4 Ilustrasi *failure link* pada node algoritma Aho-Corasick

2.1.4 Algoritma Fuzzy Matching dengan Levenshtein Distance

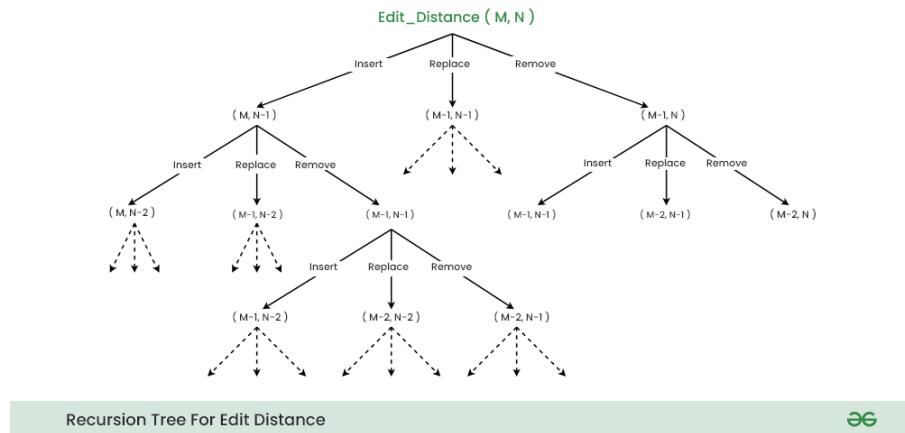
Algoritma Fuzzy *String Matching* adalah algoritma untuk pencocokan string yang mirip (dua string yang cocok sebagian tetapi tidak sama persis). Misal contoh string yang dimaksud adalah “London” dan “Londin”. Algoritma ini berupaya menentukan tingkat kedekatan antara dua string yang berbeda menggunakan “edit distance” yang menentukan seberapa dekat dua string dengan menemukan jumlah “edit” minimum yang diperlukan untuk mengubah dari satu string ke string lainnya. “Edit” yang digunakan di sini adalah *insertion*, *deletion*, dan *substitution*.

Salah satu cara untuk menghitung “edit distance” adalah Levenshtein Distance yang diusulkan oleh Vladimir Levenshtein pada tahun 1965. Levenshtein Distance mengukur jumlah minimal operasi “edit” karakter yang diperlukan untuk mengubah suatu string menjadi string lain. Semakin kecil, maka semakin mirip.



Gambar 2.5 Ilustrasi Levenshtein Distance

Levenshtein Distance umumnya menggunakan algoritma program dinamis di mana sebuah matriks dibangun untuk menyimpan jarak antara dua string, dengan tiap sel dalam matriks merepresentasikan Levenshtein Distance minimum antara prefiks pertama i karakter dari string pertama dan prefiks pertama j karakter dari string kedua.



Gambar 2.6 Ilustrasi Pohon Rekursi untuk Levenshtein Distance

2.2 Program Aplikasi

Pemilihan bahasa pemrograman Python sebagai bahasa pengembangan untuk program aplikasi adalah karena kemampuannya yang serbaguna dan keberadaan berbagai pustaka *Graphical User Interface* (GUI) yang kuat dan *cross-platform*. Pustaka seperti Tkinter, pustaka GUI standar Python memberikan kemudahan penggunaan dan cocok untuk pengembangan *interface* sederhana hingga level menengah. Sementara itu, pustaka seperti PyQt memberikan fungsionalitas yang lebih luas, tampilan yang lebih “profesional”, dan kemampuan kustomisasi yang tinggi karena didasarkan pada *framework* Qt yang komprehensif. Selain pustaka-pustaka tersebut, *framework* yang lebih modern seperti Flet menjadi salah satu opsi lain dalam pengembangan GUI di Python. Flet menawarkan paradigma unik dengan memungkinkan pembangunan aplikasi GUI *multi platform* (desktop, web, dan mobile) hanya dengan menggunakan Python, menghilangkan kebutuhan bahasa pemrograman *frontend* seperti HTML, CSS, atau JavaScript. Pendekatan ini memungkinkan pengembang untuk merancang antarmuka pengguna yang menarik dan interaktif dengan efisiensi tinggi dari satu basis kode Python, memfasilitasi *deployment* yang luas di banyak *platform*.

Prinsip dasar dari GUI melibatkan pemrograman berbasis *event-driven*, artinya sistem aplikasi merespons tindakan pengguna sebagai suatu *event* (contohnya: klik tombol, input angka, dll.); penggunaan *widgets* (elemen visual *interface* seperti tombol, label, dll.) untuk memfasilitasi interaksi pengguna terhadap aplikasi; dan manajemen *layout* untuk mengatur penempatan setiap elemen *user interface*.

BAB 3: Analisis Pemecahan Masalah

1. Langkah-langkah Pemecahan Masalah

Untuk menyelesaikan masalah untuk menemukan suatu kata yang diinginkan dalam suatu CV, diperlukan pembuatan program algoritma untuk pencocokan *pattern* atau *string*. Pengguna dapat memilih algoritma apa yang diinginkan untuk pencocokan pola atau *string matching*, kemudian program akan memberikan hasil dari proses algoritma tersebut.

Backend aplikasi dibangun menggunakan bahasa Python. Di bagian ini, dirancang empat algoritma *string matching*, yaitu algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore, Aho-Corasick, dan algoritma *fuzzy matching* dengan Levenshtein Distance. Aplikasi kemudian menggunakan kerangka pustaka Flet dalam mengembangkan GUI-nya sebagai *frontend* untuk memudahkan pengguna menggunakan aplikasi sistem ATS ini. Fitur yang dapat disediakan berupa input pencarian teks yang ingin dicari (*pattern*), pemilihan algoritma pencarian dan *fallback* ke algoritma Fuzzy, sehingga memberikan luaran hasil pencarian yang berupa *summary CV applicants*.

2. Proses Pemetaan Masalah

Proses pemetaan masalah *string matching* atau pencocokan pola diubah menjadi elemen-elemen algoritma KMP, Boyer-Moore, Aho-Corasick, dan *Fuzzy String Matching* dengan Levenshtein Distance

2.1. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma KMP digunakan untuk mencari pattern dalam teks dengan menggunakan tabel fungsi Longest Proper Suffix (LPS). Algoritma ini memanfaatkan informasi dari kecocokan karakter sebelumnya untuk menghindari pemeriksaan ulang karakter yang tidak perlu seperti metode algoritma brute force.

Pemetaan ke elemen-elemen algoritma KMP adalah sebagai berikut:

- Teks: string utama (text) panjang yang diinput sebagai tempat pencarian pola dilakukan
- Pola: substring atau pattern yang diinput pengguna untuk dicari dalam teks

- Tabel LPS: Array yang menyimpan informasi panjang prefiks terpanjang yang juga adalah sebuah sufiks untuk setiap posisi dalam pola. Dari informasi ini, digunakan untuk menentukan langkah mundur dalam pola yang perlu dilakukan saat terjadi mismatch atau ketidakcocokan.
- Status Posisi: [i] untuk string text dan [j] untuk substring pattern yang menunjukkan posisi saat ini pada masing-masing string selama proses pencocokan
- Ruang Status: Himpunan semua pasangan indeks (i, j) yang dilalui selama proses pencarian hingga pola ditemukan atau teks habis
- Ruang Solusi: Posisi awal di teks tempat pola ditemukan (apabila ditemukan)
- Proses Matching: Algoritma membandingkan tiap karakter antara teks dan pola dan apabila terjadi mismatch, menggunakan fungsi pinggiran atau boundary function pada tabel LPS untuk melompati karakter yang sudah cocok sebelumnya pada pola, sehingga mengurangi jumlah perbandingan yang diperlukan seperti brute force.

2.2. Algoritma Boyer-Moore

Algoritma Boyer-Moore (BM) adalah algoritma pattern matching yang menggunakan dua teknik, yaitu Looking-Glass Technique dan Character-Jump Technique. Teknis Looking-Glass artinya mencari pola dalam teks dengan bergerak dari kanan ke kiri dalam pola, sedangkan teknik Character-Jump digunakan ketika terjadi ketidakcocokan atau mismatch antara karakter dalam teks dengan karakter pola. Ketika terjadi mismatch, algoritma mengacu pada dua aturan, yaitu Bad Character Rule dan Good Suffix Rule untuk menentukan seberapa jauh algoritma melompati teks untuk menemukan pola berdasarkan indeks karakter terakhir muncul dalam pola (last occurrence).

Pemetaan ke elemen-elemen algoritma BM adalah sebagai berikut:

- Teks: string utama (text) panjang yang diinput sebagai tempat pencarian pola dilakukan

- Pola: substring atau pattern yang diinput pengguna untuk dicari dalam teks
- Tabel Bad Character: Sebuah peta atau map yang menyimpan indeks kemunculan terakhir setiap karakter dalam pola untuk melompati karakter teks yang tidak cocok.
- Tabel Good Suffix: Array yang menyimpan informasi tentang pergeseran atau shifting yang perlu dilakukan berdasarkan sufiks yang cocok, memungkinkan lompatan besar saat terjadi mismatch.
- Status Posisi: indeks [t] sebagai posisi awal pencarian dalam teks dan [j] sebagai indeks karakter dalam pola yang sedang dibandingkan.
- Ruang Solusi: Posisi awal di teks tempat pola ditemukan (apabila ditemukan)
- Proses Pencarian: Pencarian dilakukan dari kanan ke kiri pada pola sesuai dengan teknis Looking-Glass, dengan lompatan yang berdasarkan aturan bad character atau good suffix untuk meminimalkan perbandingan karakter seperti brute force
- Ruang Status: Himpunan semua posisi [t] yang dievaluasi selama proses pencarian

2.3. Algoritma Aho-Corasick

Algoritma Aho-Corasick adalah algoritma *pattern matching* yang mencari beberapa pola sekaligus dalam teks sebagai kamus menggunakan struktur data *Trie*, *Failure Links*, dan *Output Links*. Pemetaan ke elemen-elemen algoritma Aho-Corasick adalah sebagai berikut:

- Teks: string utama (text) panjang yang diinput sebagai tempat pencarian pola dilakukan
- Pola: *substring* atau *pattern* yang diinput pengguna untuk dicari dalam teks
- Trie: struktur data berbentuk pohon yang menyimpan semua pola atau *pattern* yang ingin dicari oleh pengguna, di mana setiap node atau

- simpul mewakili satu karakter dan jalur dari akar menuju daun mewakili pola tersebut
- Failure Link: Pointer tiap node yang menunjuk node lain dalam Trie untuk menangani kasus *mismatch*, untuk melakukan transisi cepat ke pola lain yang mungkin cocok.
 - Output: Himpunan pola yang ditemukan pada simpul tertentu di data Trie
 - Status Posisi: Simpul saat ini dalam Trie dan indeks [i] dalam teks selama proses pencarian
 - Ruang Solusi: Daftar posisi dan pola yang menunjukkan posisi di teks tempat setiap pola ditemukan
 - Proses *Matching*: Algoritma menelusuri teks karakter demi karakter, mengikuti Trie dan menggunakan *failure links* untuk menangani *mismatch*. Apabila node dengan *output* pola yang diinginkan ditemukan, pola tersebut dicatat beserta posisinya dalam teks.
 - Ruang Status: Himpunan semua simpul *Trie* yang dikunjungi selama pencarian, bersama dengan posisi teks yang sesuai dengan pola.

2.4. Algoritma Fuzzy String Matching dengan Levenshtein Distance

Algoritma *Fuzzy String Matching* menggunakan *Levenshtein Distance* untuk menemukan *substring* dalam teks yang mirip dengan pola berdasarkan ambang batas jarak tertentu (*Levenshtein Distance*). Pemetaan ke elemen-elemen algoritma ini adalah sebagai berikut:

- Teks: string utama (*text*) panjang yang diinput sebagai tempat pencarian pola dilakukan
- Pola: *substring* atau *pattern* yang diinput pengguna untuk dicari dalam teks
- Levenshtein Distance: Matriks yang menghitung jumlah operasi minimal (*deletion*, *insertion*, *substitution*) untuk mengubah satu *string* menjadi *string* lain.

- Tabel Dinamis: Matriks yang digunakan untuk menghitung Levenshtein Distance antara pola dan *substring text*.
- Max Distance: Ambang batas jarak maksimum yang diperbolehkan agar *substring* dianggap cocok dengan pola.
- Status Posisi: Posisi [i] dalam teks dan nilai Levenshtein Distance untuk setiap *substring* yang dievaluasi
- Ruang Solusi: Daftar *triplet* (posisi, *substring*, dan jarak) yang menunjukkan *substring* dalam teks yang memiliki Levenshtein Distance kurang dari atau sama dengan Max Distance terhadap pola.
- Proses *Matching*: Algoritma memeriksa setiap *substring* yang didapatkan dari pola dalam teks, menghitung Levenshtein Distance, dan mencatat *substring* tersebut dalam teks.
- Ruang Status: Himpunan semua *substring* yang dievaluasi beserta nilai *Levenshtein Distance*-nya selama proses pencarian.

3. Fitur Fungsional dan Arsitektur Aplikasi

Fitur fungsional yang disediakan dari program aplikasi kami kepada pengguna terdiri dari:

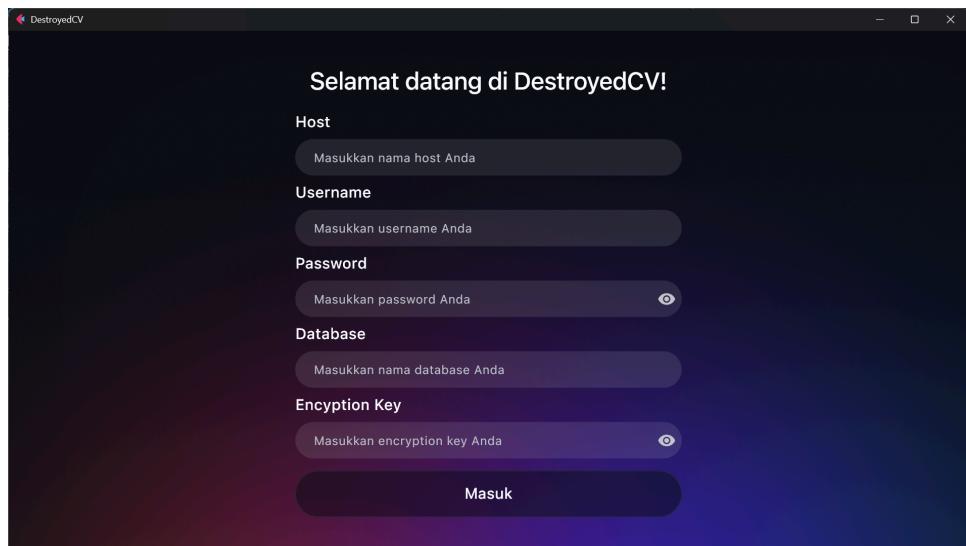
- Input Pencarian Teks: dapat berupa kata kunci atau teks berukuran besar
- Pemilihan Jumlah Top Matches: dapat memilih jumlah hasil pencarian yang diinginkan
- Pemilihan Algoritma Pencarian: Opsi yang berupa algoritma Knuth-Morris-Pratt, Boyer-Moore, dan Aho-Corasick, serta *fallback* ke algoritma Fuzzy apabila tidak ditemukan hasil pencarian sesuai input pencarian.
- Pencarian: Proses pencarian yang dilakukan sesuai dengan kriteria dari input pengguna sebelumnya
- Hasil Pencarian: Menampilkan nama pemilik CV, hasil match sesuai dengan input pencarian teks kita, serta opsi untuk melihat Summary atau CV. Disertai juga dengan statistik jumlah dan waktu pencarian.
- Summary Result: Berupa tampilan ekstraksi informasi dari CV

- View CV: Berupa tampilan file CV asli

Fitur ini dibungkus dalam suatu aplikasi GUI yang memanfaatkan framework Python berupa Flet. Framework ini memungkinkan pembuatan aplikasi web ataupun native dengan pengembangan frontend dan backend dalam Python. Arsitektur yang dibangun pada framework ini adalah sistem *frontend* dan *backend* sederhana, di mana kode sumber diorganisir sesuai dengan fungsionalitasnya. Bagian *frontend* dibangun pada direktori “src/gui”, sedangkan bagian *backend* dibangun pada direktori “src”.

Arsitektur navigasi dan routing dalam aplikasi memanfaatkan sistem *routing* dari Flet. Halaman-halaman menu pada aplikasi dibangun menggunakan Page View agar memungkinkan perubahan halaman berbasis struktur data Stack. *Routing* dilakukan dengan sederhana melalui manajemen aksi dan routing pada event tertentu, misal klik button atau menu pada *navigation bar*.

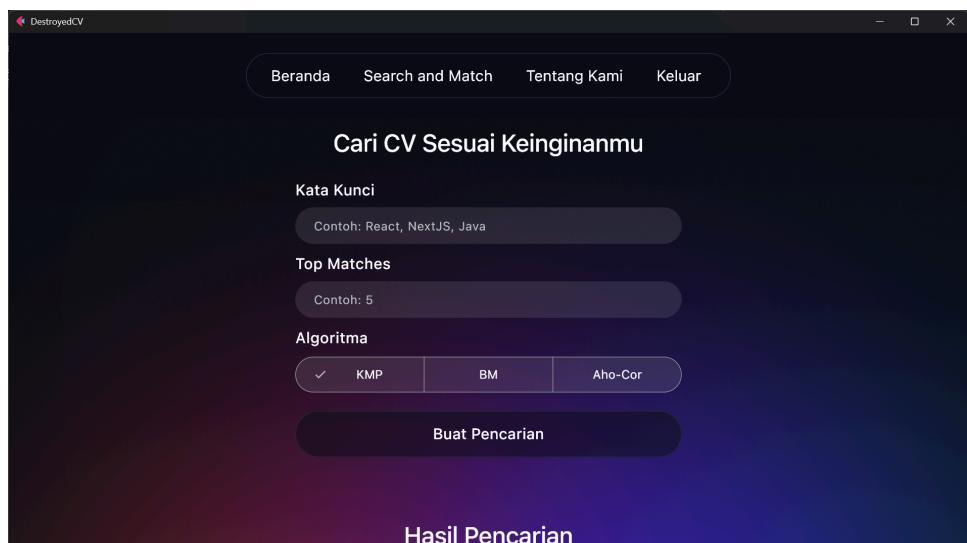
Perancangan aplikasi juga diikuti dengan perancangan UI/UX yang baik. Tampilan antarmuka diciptakan sebaik mungkin dengan juga memperhatikan pengalaman pengguna. Berikut adalah tampilan antarmuka pada aplikasi GUI kami:



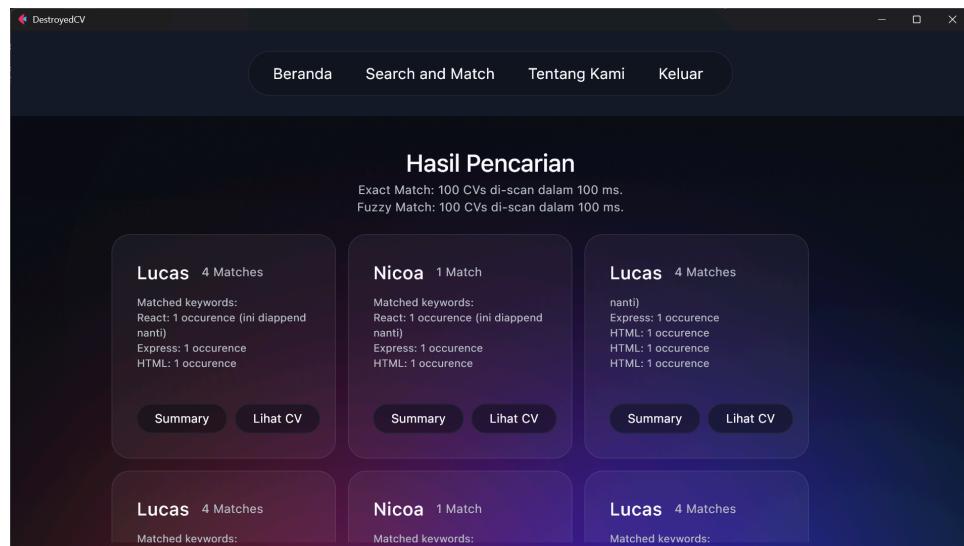
Gambar 3.1 Tampilan Halaman Login



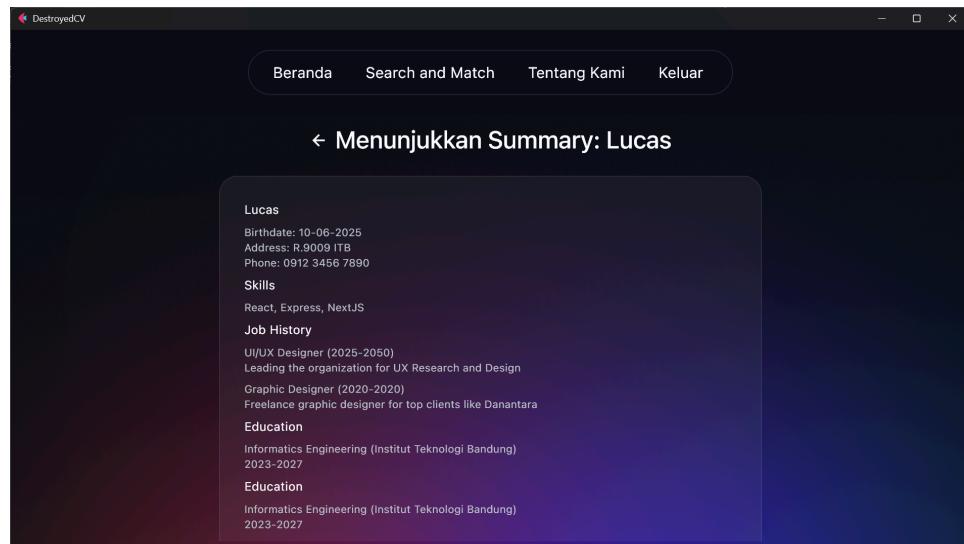
Gambar 3.2 Tampilan Halaman Beranda



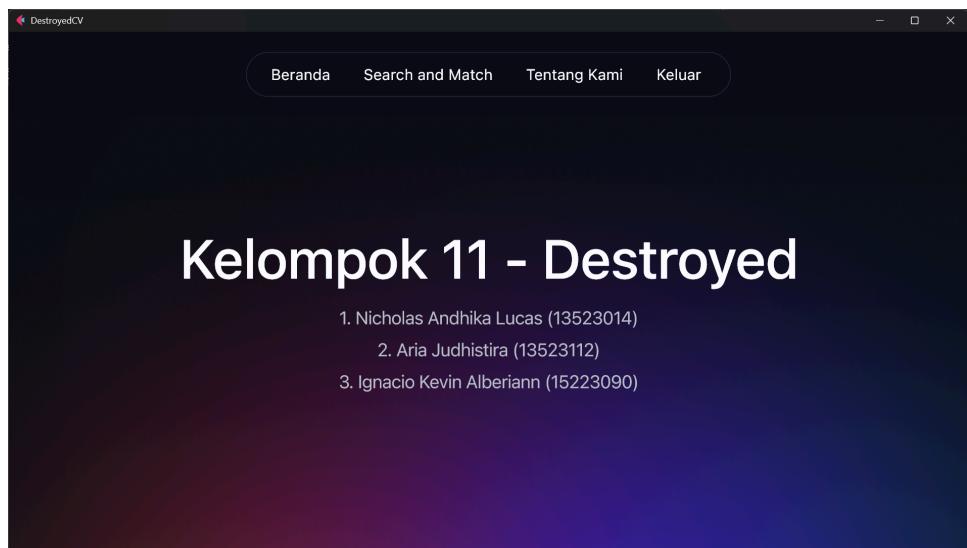
Gambar 3.3 Tampilan Halaman Pencarian



Gambar 3.4 Tampilan Halaman Hasil Pencarian



Gambar 3.2 Tampilan Halaman Summary dan CV



Gambar 3.5 Tampilan Halaman Tentang Kami

BAB 4: Implementasi dan Pengujian

4.1. Spesifikasi Teknis Program

Berikut adalah struktur program:

```
TUBES3_DESTROYED/
├── README.md
├── requirements.txt
├── data
└── src
    ├── app.py
    ├── connector.py
    ├── encryption.py
    ├── pdf_processor.py
    ├── regex_func.py
    ├── test_connector.py
    ├── algorithm
    │   ├── __init__.py
    │   ├── ac.py
    │   ├── bm.py
    │   ├── fuzzy.py
    │   └── kmp.py
    └── gui
        └── assets
            └── fonts
```

Fungsi dan prosedur yang digunakan dikelompokan berdasarkan penggunaannya dalam algoritma.

1. Implementasi Algoritma KMP (kmp.py)

Nama Fungsi	Deskripsi
compute_lps(pattern)	Fungsi ini bertanggungjawab dalam memproses <i>pattern</i> atau pola yang diinput pengguna untuk membuat array <i>Longest Proper Suffix</i> , yang juga disebut sebagai <i>failure function</i> atau <i>prefix function</i> . Fungsi menerima <i>pattern</i> dan mengembalikan array LPS yang menyimpan panjang prefiks terpanjang yang juga merupakan sufiks untuk setiap prefiks dari pola tersebut untuk menggeser pola saat terjadi <i>mismatch</i> .

KMP(text, pattern)	Fungsi ini menggunakan array LPS sebelumnya untuk mencari dan menghitung semua kemunculan <i>pattern</i> dalam teks. Fungsi ini menerima <i>text</i> dan <i>pattern</i> , lalu mengembalikan jumlah total kemunculan pola tersebut dalam teks.
--------------------	--

2. Implementasi Algoritma Boyer-Moore (bm.py)

Nama Fungsi	Deskripsi
badCharacterRule(pattern)	Fungsi ini bertanggungjawab untuk membuat peta atau <i>map</i> kemunculan karakter berdasarkan <i>bad character rule</i> . Fungsi ini menerima <i>pattern</i> dan mengembalikan sebuah <i>map</i> kamus (<i>occurrenceMap</i>). Kamus ini menyimpan indeks kemunculan terakhir dari setiap karakter dalam pola. Ketika terjadi <i>mismatch</i> , aturan ini digunakan untuk menggeser pola sehingga karakter dari teks yang tidak cocok sejajar dengan kemunculan terakhir karakter dalam pola, atau dilewati jika tidak ada karakter dalam <i>pattern</i> .
goodSuffixRule(pattern)	Fungsi ini bertanggungjawab untuk menghitung tabel pergeseran berdasarkan <i>good suffix rule</i> . Fungsi ini menerima <i>pattern</i> dan mengembalikan sebuah array geser/shift yang berisi nilai pergeseran minimal yang dibutuhkan jika sebagian sufiks dari pola telah cocok, tetapi ada <i>mismatch</i> di karakter sebelumnya. <i>Shifting</i> ini memastikan bagian sufiks yang cocok sejajar dengan kemunculan berikutnya dalam pola.
BoyerMoore(text, pattern)	Fungsi ini adalah fungsi utama yang bertanggungjawab dalam mengimplementasikan algoritma Boyer-Moore untuk mencari dan menghitung semua kemunculan <i>pattern</i> dalam <i>text</i> . Fungsi ini menerima <i>text</i> dan <i>pattern</i> , lalu mengembalikan jumlah total kemunculan <i>pattern</i> . Algoritma ini menggunakan kombinasi dari <i>bad character rule</i> dan <i>good suffix rule</i> untuk menentukan pergeseran pola yang optimal setelah setiap perbandingan karakter.

3. Implementasi Algoritma Aho-Corasick (ac.py)

Nama Kelas dan Fungsi	Deskripsi
AhoCorasick	Kelas untuk implementasi algoritma Aho-Corasick
__init__(self, patterns)	Konstruktor ini bertanggungjawab untuk membangun data <i>Trie</i> dari daftar <i>pattern</i> yang diberikan pengguna dan kemudian membuat semua <i>failure link</i> antar node. Selama pembangunan <i>trie</i> , pola-pola dimasukkan per karakter ke dalam simpul, dan simpul terakhir dari setiap pola ditandai sebagai <i>node output</i> . Pembangunan <i>failure link</i> menggunakan algoritma BFS untuk secara sistematis menentukan <i>failure link</i> setiap node.
search_detailed(self, text)	Metode ini bertanggungjawab untuk menjalankan proses <i>string matching</i> atau pencocokan pola dalam <i>text</i> . Metode ini mengembalikan daftar kamus yang berisi detail setiap kecocokan. Setiap kamus dalam list memiliki nilai “pos” (indeks awal pola ditemukan dalam teks) dan “val” (string pola yang ditemukan dalam teks).

4. Algoritma Fuzzy String Matching dengan Levenshtein Distance (fuzzy.py)

Nama Fungsi	Deskripsi
levDistance(s1, s2)	Fungsi ini bertanggungjawab untuk menghitung Levenshtein Distance antara dua <i>string</i> , “s1” dan “s2”. Fungsi ini menerima dua <i>string</i> tersebut dan mengembalikan nilai integer yang merepresentasikan jumlah minimal operasi “edit” (<i>deletion, insertion, substitution</i>) yang dibutuhkan untuk mengubah dari satu <i>string</i> menjadi <i>string</i> lainnya. Perhitungan ini menggunakan pemrograman dinamis dalam mengisi matriks untuk melacak jarak edit antar prefiks kedua <i>string</i> .
calculate_similarity(s1, s2)	Fungsi ini bertanggungjawab untuk mengubah Levenshtein Distance menjadi skor kemiripan

	dalam persentase. Fungsi ini menerima <i>s1</i> dan <i>s2</i> , menghitung Levenshtein Distance di antara keduanya, dan kemudian menormalkan jarak tersebut menjadi persentase kemiripan (0% berarti tidak ada kemiripan, 100% berarti identik sama persis).
fuzzy_search(text, pattern, min_similarity_percent=80.0)	Fungsi ini bertanggungjawab dalam melakukan pencarian “fuzzy” untuk menemukan kata-kata dalam <i>text</i> yang mirip dengan <i>pattern</i> berdasarkan ambang batas kemiripan yang ditentukan (80%). Fungsi ini menerima <i>text</i> , <i>pattern</i> , dan ambang batas tersebut. Fungsi ini terlebih dahulu memecah <i>text</i> menjadi kata-kata terpisah (tokenisasi), kemudian untuk setiap kata, dihitung skor kemiripannya menggunakan fungsi calculate_similarity. Fungsi ini mengembalikan jumlah total kata dalam teks yang skor kemiripannya dengan pola memenuhi atau melebihi batas.

5. Implementasi Manajemen Koneksi MySQL dan Enkripsi Data (connector.py)

Konstanta Global	Deskripsi
SRC_DIR	Definisi jalur direktori tempat file Python berada
ROOT_DIR	Definisi jalur direktori root proyek, satu tingkat di atas SRC_DIR untuk menemukan file lain yang relatif terhadap proyek.
SQL_PATH	Menentukan jalur relatif ke file SQL yang berisi data database, yaitu data/applicant_data.sql

Nama Kelas dan Fungsi	Deskripsi
Connector	Kelas ini berperan sebagai <i>interface</i> untuk berinteraksi dengan database MySQL, termasuk fungsionalitas enkripsi data.
__init__(self, host, user, password, database, encryption_key=None)	Konstruktor kelas Connector untuk menginisialisasi atribut yang diperlukan untuk koneksi database (seperti host, user, password,

	database). Metode ini juga menginisialisasi objek Encryption jika <code>encryption_key</code> disediakan, yang menunjukkan bahwa koneksi ini memiliki kemampuan untuk mengenkripsi/mendekripsi data profil.
<code>connect(self)</code>	Metode ini menghubungi server database MySQL. Jika koneksi berhasil, akan memeriksa apakah database yang ditentukan sudah ada. Jika ada, koneksi akan menggunakan database tersebut. Jika tidak, akan muncul <code>ValueError</code> yang mengharuskan pembuatan database terlebih dahulu.
<code>ensure_database_exists(self)</code>	Fungsi ini berperan dalam memastikan database dan menjalankan perintah SQL dari file. Ia mencoba membuat database jika belum ada. Setelah dipilih, membaca dan mengeksekusi perintah SQL dari file <code>applicant_data.sql</code> . Metode ini mengembalikan <code>boolean</code> , <code>True</code> jika berhasil atau <code>False</code> jika ada kesalahan.
<code>close(self)</code>	Fungsi yang bertanggungjawab dalam menutup koneksi database dan kursor yang aktif untuk mencegah kebocoran koneksi.
<code>get_paths_id(self)</code>	Fungsi ini mengambil data jalur CV (<code>cv_path</code>) dan ID pelamar (<code>applicant_id</code>) dari tabel <code>ApplicationDetail</code> dalam database. Metode mengembalikan daftar tuple yang berisi <code>detail_id</code> , <code>applicant_id</code> , dan <code>cv_path</code> untuk setiap entri, atau “None” jika ada kesalahan atau tidak ada koneksi.
<code>encrypt_data(self)</code>	Fungsi ini melakukan enkripsi data sensitif yang disimpan dalam tabel <code>ApplicantProfile</code> . Mengambil <code>first_name</code> , <code>last_name</code> , <code>address</code> , dan <code>phone_number</code> dari setiap profil pelamar, mengenkripsinya menggunakan objek <code>Encryption</code> , dan kemudian memperbarui database dengan versi terenkripsi. Metode mengembalikan nilai <code>boolean</code> , <code>True</code> jika enkripsi berhasil atau <code>False</code> jika tidak.
<code>decrypt_data(self)</code>	Fungsi ini melakukan dekripsi data sensitif yang sebelumnya telah dienkripsi dalam tabel <code>ApplicantProfile</code> . Mengambil data terenkripsi,

	mendekripsinya menggunakan objek Encryption, dan kemudian memperbarui database dengan versi yang sudah didekripsi. Metode mengembalikan True jika dekripsi berhasil atau False jika tidak.
get_decrypted_profile(self, applicant_id)	Fungsi ini mengambil profil pelamar tertentu berdasarkan applicant_id. lalu mendekripsi bidang-bidang sensitif secara langsung. Metode mengembalikan kamus berisi nama depan, nama belakang, alamat, dan nomor telepon yang sudah didekripsi untuk pelamar yang diminta atau “None” jika profil tidak ditemukan.

6. Implementasi Enkripsi Data

Nama Kelas dan Fungsi	Deskripsi
Encryption	Kelas ini berfungsi untuk mengenkripsi dan mendekripsi string
__init__(self, key)	Konstruktor ini menginisialisasi objek Encryption dengan “key” yang digunakan untuk enkripsi/dekripsi. Kunci diproses untuk hanya menyertakan karakter alfabet dan diubah menjadi huruf kecil (<i>lowercase</i>). ValueError akan muncul jika kunci tidak mengandung karakter alfabet.
crypt(self, text, mode)	Fungsi yang melakukan operasi enkripsi atau dekripsi. Fungsi menerima <i>text</i> dan <i>mode</i> (1 untuk enkripsi, -1 untuk dekripsi).
encrypt(self, text)	Fungsi <i>wrapper</i> dengan metode “crypt” yang memanggil self.crypt(text, 1) untuk melakukan enkripsi.
decrypt(self, text)	Fungsi <i>wrapper</i> untuk metode “crypt” yang memanggil self.crypt(text, -1) untuk melakukan dekripsi.

7. Implementasi Pemrosesan Ekstraksi Teks dari PDF (extractor.py)

Variabel Global	Deskripsi
-----------------	-----------

SRC_DIR	Definisi jalur direktori tempat file Python berada
ROOT_DIR	Definisi jalur direktori root proyek, satu tingkat di atas SRC_DIR untuk menuju file PDF

Nama Fungsi	Deskripsi
extract_text_regex(file_path)	Fungsi ini bertanggungjawab untuk mengekstrak seluruh teks dari sebuah file PDF. Fungsi menerima file_path, membuka dokumen PDF menggunakan “fitz”, mengulang setiap halaman untuk mengambil teksnya, dan mengembalikan teks gabungan yang sudah di-strip (menghilangkan spasi di awal/akhir).
extract_text_strmatching(file_path)	Fungsi ini bertanggungjawab untuk mengekstrak teks dari file PDF. Setelah ekstraksi teks dari halaman PDF, fungsi ini mengganti karakter <i>newline</i> (\n) dengan spasi dan mengganti beberapa spasi berurutan dengan satu spasi saja menggunakan “re.sub(r’s+’, ‘’, full_text)”. Fungsi mengembalikan <i>text</i> yang sudah bersih atau “None” jika terjadi kesalahan.

8. Implementasi Fungsi Pencarian CV (app.py)

Nama Kelas dan Fungsi	Deskripsi
exact_search(keywords, cv_texts, algorithm)	Fungsi ini bertanggungjawab dalam melakukan pencarian <i>exact</i> (pola identik) pada teks CV. Fungsi ini menerima daftar <i>keywords</i> dan teks dari “cv_texts”, lalu menggunakan “algorithm” yang ditentukan pengguna (KMP, BM, Aho-Corasick) untuk menemukan dan menghitung kemunculan kata kunci di setiap CV. Fungsi mengembalikan kamus hasil pencarian kecocokan <i>exact</i> dan set kata kunci yang ditemukan.
fuzzy_search_start(keywords, cv_texts, exact_search_result, percentage'75.0)	Fungsi ini bertanggungjawab untuk melanjutkan proses pencarian dengan melakukan pencarian <i>fuzzy</i> untuk kata kunci yang belum ditemukan oleh <i>exact_search</i> . Fungsi ini menerima

	<p><i>keywords</i> yang belum ditemukan, “cv_texts” dan “exact_search_result” yang sudah ada serta ambang batas kemiripan. Fungsi lalu memperbarui “exact_search_result” dengan hasil kecocokan fuzzy.</p>
start_search(keywords, algorithm, number_of_results)	Fungsi ini bertanggungjawab menerima <i>keywords</i> , “algorithm” pencarian <i>exact</i> , dan “number_of_results” yang diinginkan. Prosesnya mencakup validasi input, koneksi ke database untuk mendapatkan jalur CV, ekstraksi teks dari CV, menjalankan “exact_search”, diikuti oleh “fuzzy_search_start” untuk kata kunci tersisa. Fungsi ini kemudian memproses, mengurutkan, dan mengambil profil CV untuk hasil teratas, dan mengembalikan daftar hasil yang diurutkan beserta waktu eksekusi lama pencarian <i>exact</i> dan <i>fuzzy</i> .
get_cv_text(cv_path)	Fungsi ini bertanggungjawab dalam mengekstrak teks mentah dari file PDF yang diberikan “cv_path”. Fungsi ini mengembalikan <i>string</i> teks yang diekstrak, atau <i>None</i> jika tidak berhasil.
get_cv_info(cv_path)	Fungsi ini bertanggungjawab dalam mengekstrak dan mengurai informasi terstruktur dari CV pada format PDF. Fungsi ini menerima “cv_path”, mengekstrak teksnya menggunakan “get_cv_text”, lalu memproses untuk mendapatkan informasi seperti nama, pengalaman, atau pendidikan. Fungsi mengembalikan kamus yang berisi informasi CV yang terurai tersebut, atau <i>None</i> jika tidak berhasil.

9. Implementasi Pengujian dan Enkripsi/Dekripsi Database (test_connector.py)

Nama Kelas dan Fungsi	Deskripsi
tesst_connector(host, user, password, database, encryption_key=None)	Fungsi ini bertanggungjawab sebagai penyedia pengujian dasar untuk kelas “Connector”. Fungsi ini menginisialisasi konektor, mencoba menghubungkan ke database, mengambil profil CV, dan menutup koneksi. Fungsi ini akan

	memunculkan <i>Exception</i> jika ada langkah pengujian yang gagal, menandakan masalah koneksi atau pengambilan data.
encrypt_profiles(host, user, password, database, encryption_key)	Fungsi ini bertanggungjawab dalam mengenkripsi data profil pelamar CV yang ada dalam database. Fungsi ini menerima kredensial database dan “encryption_key”, menghubungkan ke database, lalu memanggil metode enkripsi pada “Connector” untuk mengubah data di tabel ApplicantProfile menjadi format terenkripsi.
decrypt_profiles(host, user, password, database, encryption_key)	Fungsi ini bertanggungjawab untuk mendekripsi data profil pelamar CV yang sebelumnya dienkripsi di database. Fungsi ini menerima kredensial database dan “encryption_key”, menghubungkannya ke database, lalu memanggil metode dekripsi pada kelas “Connector” untuk mengembalikan data di tabel ApplicantProfile ke format aslinya.

4.2. Tata Cara Penggunaan Program

Setelah melakukan setup aplikasi dan cara menjalankan program yang disediakan pada README file pada repository kami (lihat lampiran), berikut adalah langkah-langkah penggunaan aplikasi.

- a. Masukkan data host, username, password, database, dan encryption key untuk login ke dalam aplikasi menggunakan database tersebut. Untuk encryption key, gunakan “DESTROYEDBYstima”.
- b. Masuk ke halaman Search and Match yang dapat diakses pada navigation bar atau tombol yang ada pada halaman Beranda.
- c. Pada halaman Search and Match, masukkan kriteria pencarian berupa keyword, algoritma, dan jumlah matches yang diinginkan sebelum klik tombol Buat Pencarian.

- d. Hasil pencarian akan berada di bawah container dari form input search tersebut. Scroll ke bawah untuk melihat hasil result CV yang sesuai dengan kriteria.
- e. Untuk melihat summary atau CV dari result yang dihasilkan, klik tombol pada container result card yang menyediakan tombol tersebut.

Fitur tambahan yang diimplementasikan berupa enkripsi data, sehingga pengguna harus memasukkan encryption key di awal masuk program, serta algoritma Aho-Corasick sebagai tambahan opsi pemilihan algoritma *string matching*.

Untuk melihat referensi pada tampilan antarmuka agar mendapatkan bayangan akan langkah-langkah tersebut, dapat dilihat pada Gambar 3.1 sampai 3.5.

4.3. Hasil Pengujian

Hasil pengujian dilakukan pada 4 pencarian yang melakukan variasi pada keyword, algoritma, dan panjang keyword yang berbeda.

Keyword: banking

Input	Output
Algoritma: KMP	<p>Exact Match: 600 diproses dalam 2.1453 detik. Fuzzy Match tidak dijalankan.</p> <div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> Haikal Assy4uqi 1 Matches Matched keywords: banking: 2 occurrences (exact) Summary Lihat CV </div> <div style="text-align: center;"> MoH4mM4d NugR4h4 1 Matches Matched keywords: banking: 8 occurrences (exact) Summary Lihat CV </div> <div style="text-align: center;"> f4rhan na1f5 1 Matches Matched keywords: banking: 2 occurrences (exact) Summary Lihat CV </div> <div style="text-align: center;"> 1kHw4N 4lHaKIM 1 Matches Matched keywords: banking: 4 occurrences (exact) Summary </div> </div>

Algoritma: BM	
Algoritma: Aho-Corasick	

Keyword: design, collaborate, team, service, motivated

Input	Output
Algoritma: KMP	

Algoritma: BM	<p>Hasil Pencarian</p> <p>Exact Match: 600 diproses dalam 4.9581 detik.</p> <p>Fuzzy Match tidak dijalankan.</p> <p>H41K4L 455YAUQ1 5 Matches Matched keywords: service: 5 occurrences (exact) team: 1 occurrence (exact) collaborate: 1 occurrence (exact) Summary Lihat CV</p> <p>AHMAD RAFI 5 Matches Matched keywords: service: 1 occurrence (exact) team: 4 occurrences (exact) collaborate: 1 occurrence (exact) Summary Lihat CV</p> <p>H41K4L 455YAUQ1 5 Matches Matched keywords: team: 5 occurrences (exact) team: 1 occurrence (exact) collaborate: 1 occurrence (exact) Summary Lihat CV</p> <p>Ariel Herfrison 5 Matches Matched keywords: Summary</p> <p>Ariel Herfrison 5 Matches Matched keywords: Summary</p> <p>Ariel Herfrison 5 Matches Matched keywords: Summary</p>
Algoritma: Aho-Corasick	<p>Hasil Pencarian</p> <p>Exact Match: 600 diproses dalam 1.5129 detik.</p> <p>Fuzzy Match tidak dijalankan.</p> <p>H41K4L 455YAUQ1 5 Matches Matched keywords: motivated: 1 occurrence (exact) collaborate: 1 occurrence (exact) design: 7 occurrences (exact) Summary Lihat CV</p> <p>AHMAD RAFI 5 Matches Matched keywords: motivated: 1 occurrence (exact) team: 4 occurrences (exact) service: 1 occurrence (exact) Summary Lihat CV</p> <p>H41K4L 455YAUQ1 5 Matches Matched keywords: motivated: 1 occurrence (exact) collaborate: 1 occurrence (exact) design: 7 occurrences (exact) Summary Lihat CV</p> <p>Ariel Herfrison 5 Matches Matched keywords: Summary</p> <p>Ariel Herfrison 5 Matches Matched keywords: Summary</p> <p>Ariel Herfrison 5 Matches Matched keywords: Summary</p>

Keyword: California State University, Chemistry, Chemoreceptor, Business, million.

Top matches: 10

Input	Output
Algoritma: KMP	<p>Hasil Pencarian</p> <p>Exact Match: 600 diproses dalam 3.9766 detik.</p> <p>Fuzzy Match dijalankan dalam 152.9517 detik.</p> <p>M0h4mM4D NUGRAHA 3 Matches Matched keywords: Chemistry: 17 occurrences (fuzzy) Business: 9 occurrences (fuzzy) Summary Lihat CV</p> <p>FARH4N NAFIS 2 Matches Matched keywords: million.: 1 occurrence (exact) Business: 15 occurrences (fuzzy) Summary Lihat CV</p> <p>4I4nd MuL14 2 Matches Matched keywords: million.: 2 occurrences (exact) Summary Lihat CV</p> <p>4I4nd MuL14 2 Matches Matched keywords: million.: 2 occurrences (exact) Summary Lihat CV</p> <p>ARI3L h3rfri50n 2 Matches Matched keywords: Chemistry: 2 occurrences (fuzzy) Summary Lihat CV</p>

Algoritma: BM	<p style="text-align: center;">Hasil Pencarian</p> <p>Exact Match: 600 diproses dalam 1.1015 detik. Fuzzy Match dijalankan dalam 80.7230 detik.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Profile</th> <th>Matches</th> </tr> </thead> <tbody> <tr><td>M0h4mM4D NUGRAHA</td><td>3 Matches</td></tr> <tr><td>FARH4N NAFIS</td><td>2 Matches</td></tr> <tr><td>4l4nd MuL14</td><td>2 Matches</td></tr> <tr><td>ARI3L h3rfri50n</td><td>2 Matches</td></tr> <tr><td>4riel HerfrisOn</td><td>2 Matches</td></tr> </tbody> </table>	Profile	Matches	M0h4mM4D NUGRAHA	3 Matches	FARH4N NAFIS	2 Matches	4l4nd MuL14	2 Matches	ARI3L h3rfri50n	2 Matches	4riel HerfrisOn	2 Matches
Profile	Matches												
M0h4mM4D NUGRAHA	3 Matches												
FARH4N NAFIS	2 Matches												
4l4nd MuL14	2 Matches												
ARI3L h3rfri50n	2 Matches												
4riel HerfrisOn	2 Matches												
Algoritma: Aho-Corasick	<p style="text-align: center;">Hasil Pencarian</p> <p>Exact Match: 600 diproses dalam 1.1160 detik. Fuzzy Match dijalankan dalam 89.6796 detik.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Profile</th> <th>Matches</th> </tr> </thead> <tbody> <tr><td>M0h4mM4D NUGRAHA</td><td>3 Matches</td></tr> <tr><td>FARH4N NAFIS</td><td>2 Matches</td></tr> <tr><td>4l4nd MuL14</td><td>2 Matches</td></tr> <tr><td>ARI3L h3rfri50n</td><td>2 Matches</td></tr> <tr><td>4riel HerfrisOn</td><td>2 Matches</td></tr> </tbody> </table>	Profile	Matches	M0h4mM4D NUGRAHA	3 Matches	FARH4N NAFIS	2 Matches	4l4nd MuL14	2 Matches	ARI3L h3rfri50n	2 Matches	4riel HerfrisOn	2 Matches
Profile	Matches												
M0h4mM4D NUGRAHA	3 Matches												
FARH4N NAFIS	2 Matches												
4l4nd MuL14	2 Matches												
ARI3L h3rfri50n	2 Matches												
4riel HerfrisOn	2 Matches												

4.4. Analisis Hasil Pengujian

Berdasarkan pengujian pertama yang dilakukan, yaitu keyword “banking” saja, algoritma BM melakukan pencarian yang tercepat dengan waktu 0.3 detik, diikuti dengan algoritma Aho-Corasick yaitu selama 0.9 detik. Algoritma KMP memakan waktu yang paling lama, yaitu 2.1 detik. Namun, urutan hasil pencarian ini berbeda di pengujian kedua yang menggunakan lebih banyak keyword.

Di pencarian kedua dengan keyword “design, collaborate, team, service, motivated”, algoritma tercepat adalah Aho-Corasick dengan lama pencarian 1.5 detik, diikuti dengan algoritma KMP selama 3.6 detik, dan terakhir BM selama 4.9 detik.

Pencarian terakhir yang menggunakan keyword yang lebih spesifik dan sulit, “California State University, Chemistry, Chemoreceptor, Business, million”, akhirnya memaksa menggunakan fuzzy match untuk mendapatkan hasil CV yang memiliki kemiripan dari keyword yang dimasukkan. Algoritma BM dan Aho-Corasick

sama-sama cepat dengan waktu pencarian selama 1.1 detik, sedangkan algoritma KMP lebih lambat, selama 3.9 detik.

Berdasarkan hasil pengujian ini, dapat disimpulkan bahwa Aho-Corasick relatif merupakan algoritma yang stabil untuk digunakan dalam berbagai jenis pencarian. Sesuai dengan karakteristiknya, ia cepat melakukan pemrosesan untuk pola banyak, sehingga merupakan algoritma paling cepat dibanding BM dan KMP untuk pengujian dengan kata kunci yang banyak. Sedangkan, untuk algoritma KMP, ia memiliki kemampuan pencarian yang linear dan akan semakin melambat untuk pencarian dengan kata kunci atau variasi karakter yang banyak. Terakhir, algoritma KMP memiliki potensi kecepatan pencarian yang lebih tinggi karena bisa melompati beberapa pencarian yang tidak dibutuhkan. Namun, pada kasus spesifik, misalnya untuk kata kunci yang lebih banyak dan umum, keuntungan ini bisa hilang.

BAB 5: Kesimpulan, Saran, dan Refleksi

5.1 Kesimpulan

Tugas Besar III Strategi Algoritma menyajikan permasalahan pencarian pola atau *string matching* dalam membangun sistem ATS (*Applicant Tracking System*) berbasis CV Digital. Proses pencarian/pencocokan pola dalam teks menggunakan algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore, Aho-Corasick, dan Fuzzy String Matching dengan Levenshtein Distance. Aplikasi dibuat menggunakan basis data berbasis SQL dan fungsi enkripsi-dekripsi pada data profil *applicant*. Aplikasi berhasil dalam menampilkan rangkuman CV *applicant*, menjadi bukti bahwa aplikasi berhasil mengekstrak informasi penting dari CV *applicant* menggunakan Regular Expression (Regex) dan algoritma pencarian/pencocokan pola.. Keseluruhan program diimplementasikan dalam bahasa pemrograman Python dengan GUI dibuat dalam kerangka pustaka Flet.

Segala macam bentuk penggeraan, seperti dasar teori, hasil pembentukan *source code*, pengimplementasian fungsi, serta tampilan layar hasil kerja kelompok telah didokumentasikan secara terstruktur dalam laporan tugas besar III ini. Meski jauh dari sempurna, kelompok Destroyed berhasil membuat program aplikasi sistem ATS (*Applicant Tracking System*) dengan memanfaatkan algoritma *pattern matching* sebagai hasil akhir dari Tugas besar III Strategi Algoritma.

5.2 Saran

Kelompok Destroyed memahami betul bahwa hasil akhir yang dibentuk ini sangat jauh dari kata maksimal dan terdapat beberapa perbaikan yang dapat diusahakan lebih lanjut sehingga program menjadi lebih efisien dan efektif. Sangat disayangkan waktu pelaksanaan penggeraan Tugas Besar III ini bersamaan dengan pekan Ujian Akhir Semester (UAS) sehingga hasil dari penggeraan kelompok Destroyed dalam tugas ini tidaklah sempurna untuk mencapai hasil terbaik.

5.3 Refleksi

Kelompok Destroyed mengucapkan terima kasih sebesar-besarnya kepada Bu Ulfa dan Pak Rinaldi selaku dosen pengampu mata kuliah Strategi Algoritma pada kampus Ganesha serta seluruh asisten yang telah memberikan perannya dalam pelaksanaan Tugas Besar Ketiga atas kesempatan yang telah diberikan kepada kelompok untuk mengerjakan Tugas Besar ketiga mata kuliah Strategi Algoritma ini. Melalui tugas ini, kelompok telah mendapatkan berbagai bentuk ilmu-ilmu dalam lingkup keinformatikaan, pengalaman dalam membentuk proyek informatika, serta latihan keterampilan-keterampilan yang relevan bagi seorang mahasiswa informatika dan non-informatika, seperti kemampuan bekerja sama dan berkomunikasi dalam kelompok, kemampuan menggunakan *git* sebagai media kolaborasi, kemampuan *debugging* dan melakukan *handle* terhadap *error* dalam kode pemrograman, dan keterampilan relevan lainnya yang dapat membantu di masa depan. Kiranya apa yang didapatkan melalui penggerjaan tugas besar ini menjadi bekal yang baik bagi anggota kelompok untuk melangsungkan keseharian perkuliahan di jurusan Teknik Informatika, jurusan Arsitektur, HMPS, ITB, dan wadah lainnya ke depannya.

Lampiran

1. Tautan Repository

Repository: https://github.com/TukangLas21/Tubes3_Destroyed

2. Tabel Checklist

Tabel 1. Tabel Checklist

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

Daftar Pustaka

blog.devgenius.io. (2025). *String Matching KMP*. Diakses pada 14 Juni 2025, dari <https://blog.devgenius.io/string-matching-kmp-ccad1ffbeb11>

CP-Algorithms. (2025). *Aho-Corasick*. Diakses pada 15 Juni 2025, dari https://cp-algorithms.com/string/aho_corasick.html

DataCamp. (2025). *Fuzzy String Matching in Python Tutorial*. Diakses pada 15 Juni 2025, dari <https://www.datacamp.com/tutorial/fuzzy-string-python>

Flet Dev. (2025). *Flet Docs*. Diakses pada 15 Juni 2025, dari <https://flet.dev/docs/>

GeeksforGeeks. (2025). *KMP Algorithm for Pattern Searching*. Diakses pada 14 Juni 2025, dari <https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/>

GeeksforGeeks. (2024). *String Manipulation Techniques*. Diakses pada 15 Juni 2025, dari <https://www.geeksforgeeks.org/dsa/edit-distance-dp-5/>

Informatika.stei.itb.ac.id. (2025). *Pencocokan string*. Diakses pada 14 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)

Owen, Elvan . (2015). *Aho – Corasick Algorithm in Pattern Matching*. Informatika.stei.itb.ac.id. Diakses pada 15 Juni 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF2_21_Strategi_Algoritma_2015_032.pdf

Python Wiki. (2025). *GuiProgramming*. Diakses pada 15 Juni 2025, dari <https://wiki.python.org/moin/GuiProgramming>

TutorialsPoint. (2025). *Boyer-Moore Algorithm*. Diakses pada 14 Juni 2025, dari https://www.tutorialspoint.com/data_structures_algorithms/boyer_moore_algorithm.htm

Wikipedia. (1977). *Knuth–Morris–Pratt algorithm*. Diakses pada 14 Juni 2025, dari https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm