

Laporan Tugas Besar 3
IF2211 Strategi Algoritma
Pemanfaatan Pattern Matching untuk Membangun Sistem ATS
(Applicant Tracking System) Berbasis CV Digital



Dipersiapkan oleh:

Farrel Jabarr	10122057
Aramazaya	13523082
Athian Nugraha Muarajuang	13523106

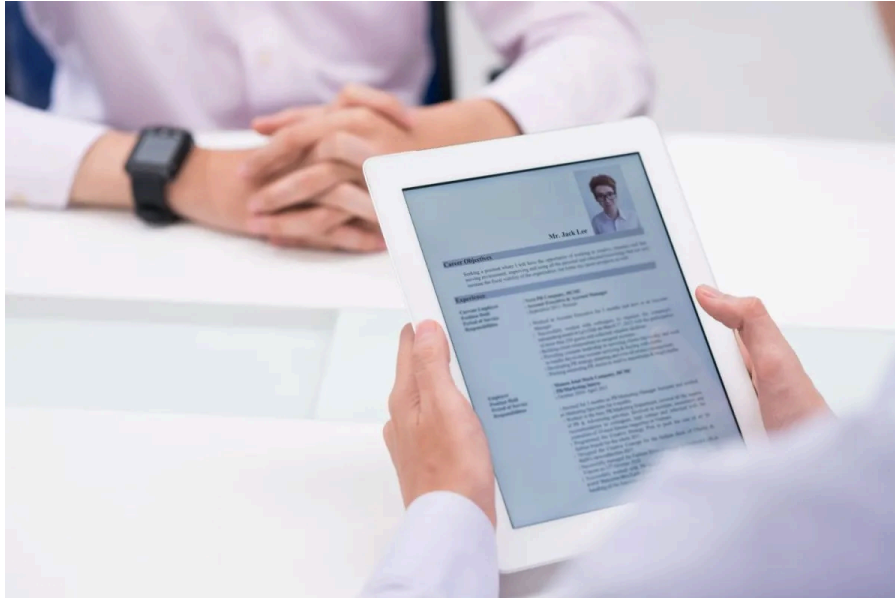
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024/2025

DAFTAR ISI

DAFTAR ISI.....	1
BAB 1	
DESKRIPSI TUGAS.....	1
Penjelasan Implementasi.....	3
Penggunaan Program.....	5
BAB 2	
LANDASAN TEORI.....	7
2.1. Knuth-Morris-Pratt (KMP).....	7
2.2. Boyer-Moore (BM).....	8
2.3. Aho-Corasick (AC).....	10
2.4 Levenshtein Distance.....	11
2.5. Aplikasi CV Analyzer.....	11
BAB 3	
ANALISIS PEMECAHAN MASALAH.....	13
3.1. Langkah-Langkah Pemecahan Masalah.....	13
3.2. Proses Pemetaan Masalah menjadi Elemen-Element Algoritma KMP dan BM.....	13
3.3. Fitur fungsional dan Arsitektur Aplikasi GUI.....	14
3.3.1. Fitur Fungsional.....	14
3.3.2. Arsitektur Aplikasi GUI.....	14
3.4. Ilustrasi Kasus.....	15
3.4.1. Ilustrasi Kasus dengan Algoritma KMP.....	15
3.4.2. Ilustrasi Kasus dengan Algoritma BM.....	16
3.4.3. Ilustrasi Kasus dengan Algoritma AC.....	16
BAB 4	
IMPLEMENTASI DAN PENGUJIAN.....	18
4.1. Spesifikasi Teknis Program.....	18
4.2. Tata Cara Penggunaan Program.....	19
4.3. Hasil Pengujian.....	20
4.3.1. Test Case 1.....	20
4.3.2. Test Case 2.....	20
4.3.3. Test Case 3.....	21
BAB 5	
KESIMPULAN DAN SARAN.....	22
5.1. Kesimpulan.....	22
5.2. Saran.....	22
LAMPIRAN.....	23
REFERENSI.....	24

BAB 1

DESKRIPSI TUGAS



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

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

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

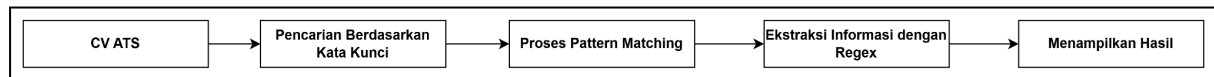
Pattern matching adalah teknik untuk menemukan dan mencocokkan pola tertentu dalam teks. Dalam konteks ini, algoritma Boyer-Moore dan Knuth-Morris-Pratt (KMP) sering digunakan karena keduanya menawarkan efisiensi tinggi untuk pencarian teks di dokumen besar. Algoritma ini memungkinkan sistem ATS untuk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

Di dalam Tugas Besar 3 ini, Anda diminta untuk mengimplementasikan sistem yang dapat melakukan deteksi informasi pelamar berbasis dokumen CV digital. Metode yang akan digunakan untuk melakukan deteksi pola dalam CV adalah Boyer-Moore dan Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas kandidat

melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali profil pelamar secara lengkap hanya dengan menggunakan CV digital.

Penjelasan Implementasi

Dalam tugas ini, Anda akan mengembangkan sebuah sistem ATS (Applicant Tracking System) berbasis CV Digital dengan memanfaatkan teknik Pattern Matching. Implementasi sistem ini akan menggunakan algoritma Boyer-Moore dan Knuth-Morris-Pratt (*Aho-Corasick* apabila mengerjakan bonus) untuk menganalisis dan mencocokkan pola dalam dokumen CV digital, sesuai dengan konsep yang telah dipelajari dalam materi dan slide perkuliahan.




Gambar 2. Skema Implementasi *Applicant Tracking System*

Sistem ini bertujuan untuk mencocokkan kata kunci dari user terhadap isi CV pelamar kerja dengan pendekatan pattern matching menggunakan algoritma KMP (Knuth-Morris-Pratt) atau BM (Boyer-Moore). Semua proses dilakukan secara in-memory, tanpa menyimpan hasil pencarian—hanya data mentah (raw) CV yang disimpan. Pengguna (HR atau rekruter) akan memberikan input berupa daftar kata kunci yang ingin dicari (misalnya: "python", "react", dan "sql") serta jumlah CV yang ingin ditampilkan (misalnya Top 10 matches). Setiap file CV dalam format PDF akan dikonversi menjadi satu string panjang yang memuat seluruh teks dari dokumen tersebut. Proses konversi ini bertujuan untuk mempermudah pencocokan pola menggunakan algoritma string matching, sehingga setiap keyword dapat dicari secara efisien dalam satu representasi data linear.

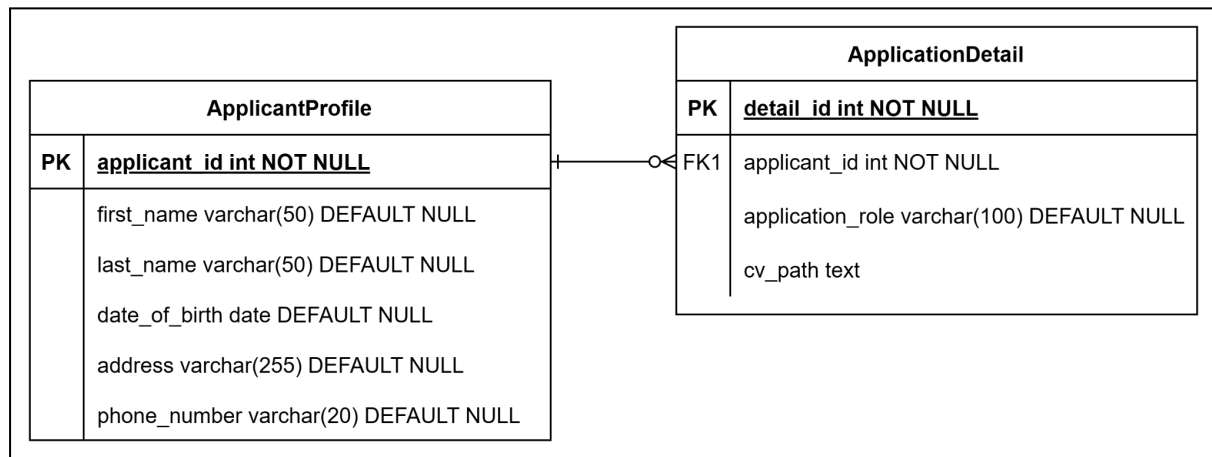
Untuk memberikan pemahaman yang lebih konkret, berikut disajikan contoh kasus penerapan sistem CV ATS beserta prosesnya dan contoh output yang dihasilkan. Dataset yang digunakan dalam contoh ini merupakan dataset CV ATS yang tercantum pada bagian referensi.

Tabel 1. Hasil ekstraksi teks dari CV ATS

CV ATS	Ekstraksi Text untuk Regex	Ekstraksi Text untuk <i>Pattern Matching</i> (KMP & BM)
 10276858.pdf	Ekstraksi Text Regex.txt	Ekstraksi Text Pattern Matching.txt

Pada tahap implementasi ini, setiap CV yang telah dikonversi menjadi string panjang untuk mempermudah proses pencocokan. Representasi ini menjadi dasar dalam mencari CV yang paling relevan dengan kata kunci yang dimasukkan oleh pengguna. Proses pencarian dilakukan dengan menggunakan algoritma pencocokan string Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) untuk menemukan CV yang memiliki kemiripan tertinggi dengan kebutuhan yang ditentukan. Apabila tidak ditemukan satupun CV dalam basis data yang memiliki kecocokan kata kunci secara exact match menggunakan algoritma KMP maupun Boyer-Moore, maka sistem akan mencari CV yang paling mirip berdasarkan tingkat kemiripan di atas ambang batas tertentu (threshold). Hal ini mempertimbangkan

kemungkinan adanya kesalahan pengetikan (typo) oleh pengguna atau HR saat memasukkan kata kunci. Anda diberikan **keleluasaan untuk menentukan nilai ambang batas persentase kemiripan tersebut**, dengan syarat dilakukan pengujian terlebih dahulu untuk menemukan nilai tuning yang optimal dan **dijelaskan secara rinci dalam laporan**. Metode perhitungan tingkat kemiripan harus diterapkan menggunakan algoritma **Levenshtein Distance**.



Gambar 3. Skema basis data CV ATS

Dalam skema basis data ini, tabel **ApplicantProfile** menyimpan informasi pribadi pelamar, sedangkan tabel **ApplicationDetail** menyimpan detail aplikasi yang diajukan oleh pelamar tersebut. Relasi antara tabel **ApplicantProfile** dan **ApplicationDetail** adalah one-to-many, karena seorang pelamar dapat mengajukan lamaran untuk beberapa posisi dalam perusahaan yang sama, atau bahkan perusahaan yang berbeda. Setiap lamaran mungkin memerlukan dokumen yang berbeda, seperti CV yang telah disesuaikan untuk peran tertentu.

Untuk keperluan pengembangan awal, basis data silahkan **di-seeding secara mandiri** menggunakan data simulasi. Mendekati tenggat waktu pengumpulan tugas, asisten akan menyediakan seeding resmi yang akan digunakan untuk Demo Tugas Besar.

Atribut **cv_path** pada tabel **ApplicationDetail** digunakan untuk menyimpan lokasi berkas CV digital pelamar di dalam repositori sistem. Lokasi penyimpanan mengikuti struktur folder di direktori **data/**, sebagaimana dijelaskan dalam struktur *repository* pada bagian [pengumpulan tugas](#). Berkas CV yang tersimpan akan dianalisis oleh sistem ATS (Applicant Tracking System) yang dikembangkan dalam Tugas Besar ini.

Penggunaan Program

CV Analyzer App

Keywords :
React, Express, HTML

Search Algorithm:
KMP ☒ BM

Top Matches:
3

Search

Results
100 CVs scanned in 100ms

Farhan	4 matches	Aland	1 match	Ariel	1 match
Matched keywords:					
1. React: 1 occurrence					
2. Express: 2 occurrences					
3. HTML: 1 occurrence					
Summary View CV		Summary View CV		Summary View CV	

Gambar 4. Contoh Antarmuka Program (Halaman *Home*)

CV Summary

Farhan

Birthdate: 05-19-2025
Address: Masjid Salman ITB
Phone: 0812 3456 7890

Skills:
React Express HTML

Job History:
CTO
2003-2004
Leading the organization's technology strategies

Education:
Informatics Engineering (Institut Teknologi Bandung)
2022-2026

Gambar 5. Contoh Antarmuka Program (Halaman *Summary*)

Anda diperbolehkan menambahkan elemen tambahan seperti gambar, logo, atau komponen visual lainnya. Desain antarmuka untuk aplikasi desktop **tidak wajib mengikuti tata letak persis** seperti contoh yang diberikan, namun harus dibuat semenarik mungkin, serta tetap mencakup seluruh **komponen wajib yang telah ditentukan**:

- Judul Aplikasi
- Kolom input kata kunci memungkinkan pengguna memasukkan satu atau lebih *keyword*, yang dipisahkan dengan koma, seperti contoh: React, Express, HTML.
- Tombol toggle memungkinkan pengguna memilih salah satu dari dua algoritma pencarian, yaitu KMP atau BM, dengan hanya satu algoritma yang bisa dipilih pada satu waktu.
- *Top Matches Selector* digunakan untuk memilih jumlah CV teratas yang ingin ditampilkan berdasarkan hasil pencocokan.

- *Search Button* digunakan untuk memulai proses pencarian. Diletakkan secara mencolok di bawah *input field*.
- *Summary Result Section* berisi informasi waktu eksekusi pencarian untuk kedua tipe matching yang dilakukan (*exact match* dengan KMP/BM dan *fuzzy match* dengan Levenshtein Distance), misalnya: “Exact Match: 100 CVs scanned in 100ms.\n Fuzzy Match: 100 CVs scanned in 101ms.”
- *Container* hasil pencarian atau kartu CV digunakan untuk menampilkan data hasil pencocokan berdasarkan keyword yang sesuai. Setiap kartu memuat informasi seperti nama kandidat, jumlah kecocokan yang dihitung dari jumlah keyword yang ditemukan, serta daftar kata kunci yang cocok beserta frekuensi kemunculannya. Selain itu, tersedia dua tombol aksi: tombol *Summary* untuk menampilkan ekstraksi informasi dari CV, serta tombol *View CV* yang memungkinkan pengguna melihat langsung file CV asli.

Secara umum, berikut adalah cara umum penggunaan program:

1. Pengguna memasukkan kata kunci pencarian.
 2. Memilih algoritma pencocokan: KMP atau BM.
 3. Menentukan jumlah hasil yang ingin ditampilkan.
 4. Menekan tombol Search.
- Sistem menampilkan daftar CV yang paling relevan, disertai tombol untuk melihat detail (*Summary*) atau CV asli (*View CV*).

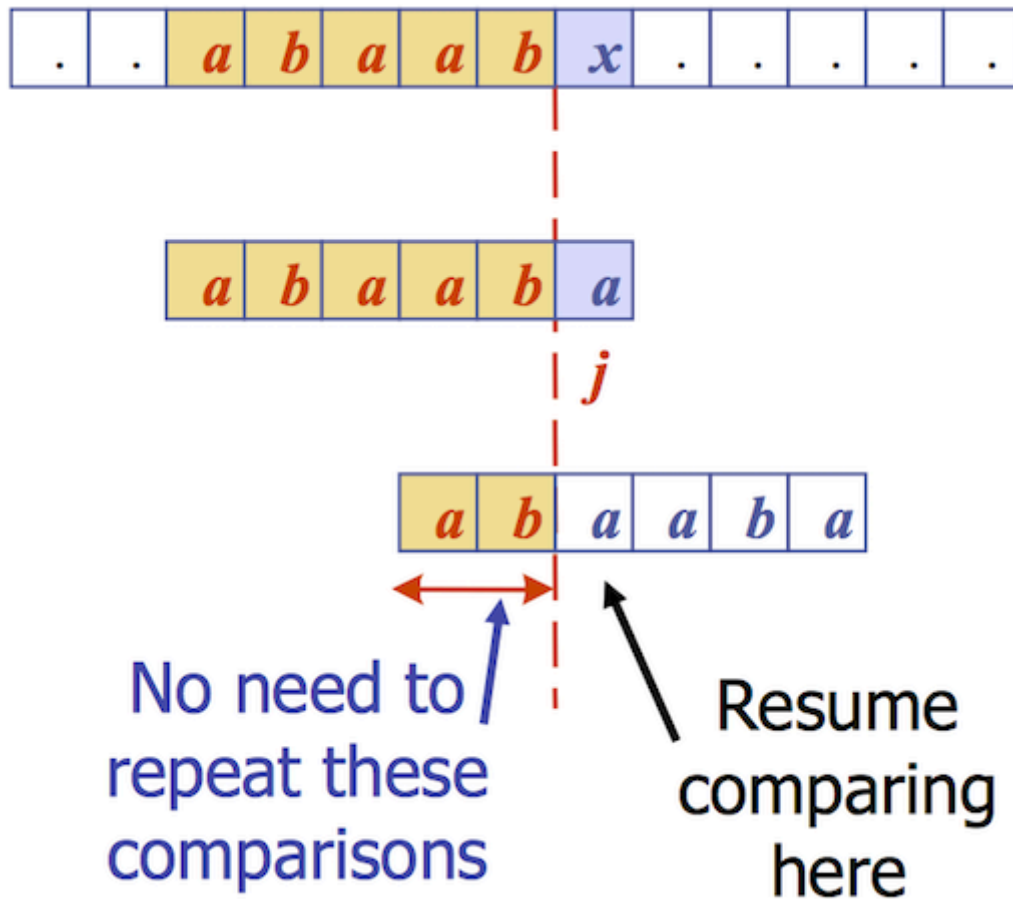
BAB 2

LANDASAN TEORI

2.1. Knuth-Morris-Pratt (KMP)

Seiring berkembangnya teknologi pattern matching, seorang profesor di stanford, Donald E. Knuth, mengemukakan cara untuk mencari sebuah pola pada sebuah string yang lebih efisien dari pencocokan Brute-force. Algoritma ini disebut sebagai the Knuth-Morris-Pratt pattern matching algorithm (KMP).

Untuk meningkatkan waktu pencarian, sebuah program dapat melakukan skip terhadap karakter-karakter yang bisa dianggap tidak perlu dilakukan pencocokan lagi. Pada algoritma KMP, hal ini dilakukan dengan mencari prefix terbesar yang juga merupakan suffix pada potongan pattern tersebut hingga posisi dimana ketidakcocokan terjadi. Jika terjadi ketidakcocokan pada index j ($T[i] \neq P[j]$), maka akan dicari prefix terbesar di $P[0-j]$ yang sesuai dengan suffix terbesar di $P[1-j]$.



Gambar 2. Pencocokan KMP

(sumber : Salindia perkuliahan UNSW COMP2521

<https://cgi.cse.unsw.edu.au/~cs2521/19T1/lects/week09a/#s9>)

Jika melihat ke contoh, pengecekan terhadap `ab` awal tidak dilakukan lagi karena sudah pasti match dengan `ab` yang berada pada posisi 3-4 (0-indexing).

Untuk mempermudah pergeseran pattern, digunakan *failure function* atau bisa juga disebut *border function*. Fungsi ini mengembalikan jumlah pergeseran yang perlu dilakukan oleh pattern setelah melihat suffix dan prefix. Cara kerja *failure function* adalah dengan mengkalkulasi jumlah *shift* (pergeseran) untuk semua posisi index dimana pencocokan terhenti.

Example: $P = \text{abaaba}$

j	0	1	2	3	4	5
P_j	a	b	a	a	b	a
$F(j)$	0	0	1	1	2	3

Gambar 3. Failure Function untuk pattern “abaaba”

(sumber : Salindia perkuliahan UNSW COMP2521

<https://cgi.cse.unsw.edu.au/~cs2521/19T1/lects/week09a/#s9>)

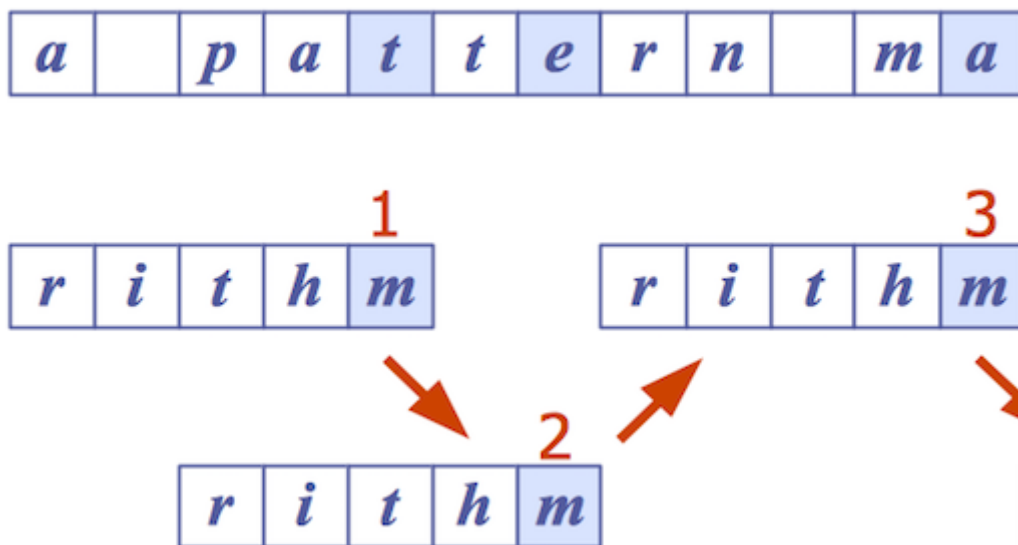
Bisa dilihat ketika index 2, didapat $F(2) = 1$ dikarenakan ‘a’ merupakan prefix dan juga suffix pada $P[0-2]$. Begitu juga dengan index 4 dimana ‘ab’ merupakan suffix dan juga prefix terbesar ($P[1-2] == P[3-4]$). Sehingga ketika terjadi ketidakcocokan pada posisi j (0-indexing), program perlu melakukan shift sebesar $F(j-1)$.

Dengan menggunakan algoritma ini, kompleksitas waktu yang didapatkan adalah $O(m+n)$ dimana $O(m)$ untuk pembuatan *failure function* dan $O(n)$ untuk pencocokan. Algoritma ini kurang pas untuk pencocokan dengan alfabet yang luas karena kemungkinan untuk *mismatch* makin tinggi.

2.2. Boyer-Moore (BM)

Selanjutnya adalah algoritma Boyer-Moore. Algoritma ini merupakan algoritma pattern matching lain yang berbeda dari KMP. Dengan Boyer-Moore, dilakukan shift berdasarkan kemunculan terakhir char yang tidak cocok.

Algoritma Boyer-Moore memiliki dasar pada dua heuristic yaitu *Looking-Glass Heuristic* dan *Character-jump Heuristic*. *Looking-glass Heuristic* artinya melakukan komparasi secara terbalik dari indeks paling belakang pattern. *Character-jump* merupakan cara melakukan *shift* dimana jika terjadi *mismatch* pada karakter $i + j$ pada teks, maka dicari kemunculan dengan indeks terbesar dari karakter tersebut pada pattern dan dilakukan *shift* sehingga $i+j$ sejajar dengan kemunculan tersebut. Jika tidak ditemukan, maka akan dilakukan *shift* sehingga $P[0]$ sejajar dengan $T[i+1]$.



Gambar 4. Cara kerja Boyer-Moore

(sumber : Salindia perkuliahan UNSW COMP2521

<https://cgi.cse.unsw.edu.au/~cs2521/19T1/lects/week09a/#s14>)

Jika dilihat di contoh, program mencocokkan 'm' dengan 't' namun karena tidak sesuai, program menggeser pattern hingga 't' di teks sesuai dengan 't' di pattern "rithm". Kemudian program mencocokkan 'm' dengan 'e', namun karena tidak terdapat 'e' di pattern, *shift* dilakukan hingga $P[0]$ sesuai dengan $T[i+1]$.

Untuk mempermudah perhitungan Boyer-Moore, dapat dibuat sebuah fungsi *Last-Occurrence* atau $L()$ function. Fungsi ini menemukan semua character pada teks dengan kemunculan terakhirnya di pattern. Jika tidak ditemukan maka dijadikan -1.

Example: $\Sigma = \{a, b, c, d\}$, $P = acab$

c	a	b	c	d
$L(c)$	2	3	1	-1

Gambar 5. $L()$ function untuk acab dengan kumpulan karakter $\{a,b,c,d\}$

(sumber : Salindia perkuliahan UNSW COMP2521

<https://cgi.cse.unsw.edu.au/~cs2521/19T1/lects/week09a/#s14>)

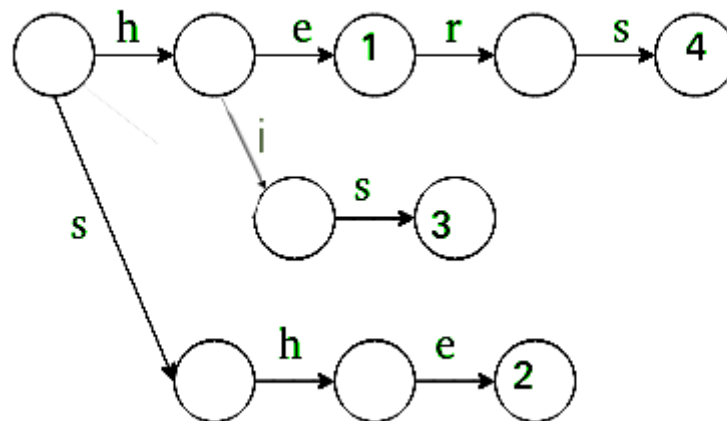
Dengan menggunakan tabel tersebut, dapat dicari *shift* menggunakan panjang pattern - $L(c)$. Algoritma ini berjalan pada $O(nm+s)$ dimana n adalah panjang teks, m adalah panjang pattern, dan s adalah jumlah alfabet yang digunakan.

2.3. Aho-Corasick (AC)

Ketika memerlukan pencarian berbagai pattern pada sebuah teks, penggunaan algoritma linear seperti KMP dan Boyer-Moore akan sangat tidak efisien karena harus dilakukan satu persatu untuk setiap kata sehingga menghasilkan kompleksitas $O(n*k+m)$ dimana n adalah panjang teks, k adalah jumlah pattern, dan m adalah panjang total pattern. Algoritma Aho-Corasick ada untuk menyelesaikan masalah tersebut. Algoritma ini memiliki kompleksitas $O(n+m+z)$ dimana z adalah jumlah total kemunculan pattern-pattern di teks.

Algoritma ini bekerja dengan pertama membuat sebuah automaton yang memiliki fungsi go to ($g[][]$), failure($F[]$), dan output($O[]$). Program membuat trie (*keyword tree*) yang akan digunakan untuk fungsi go to.

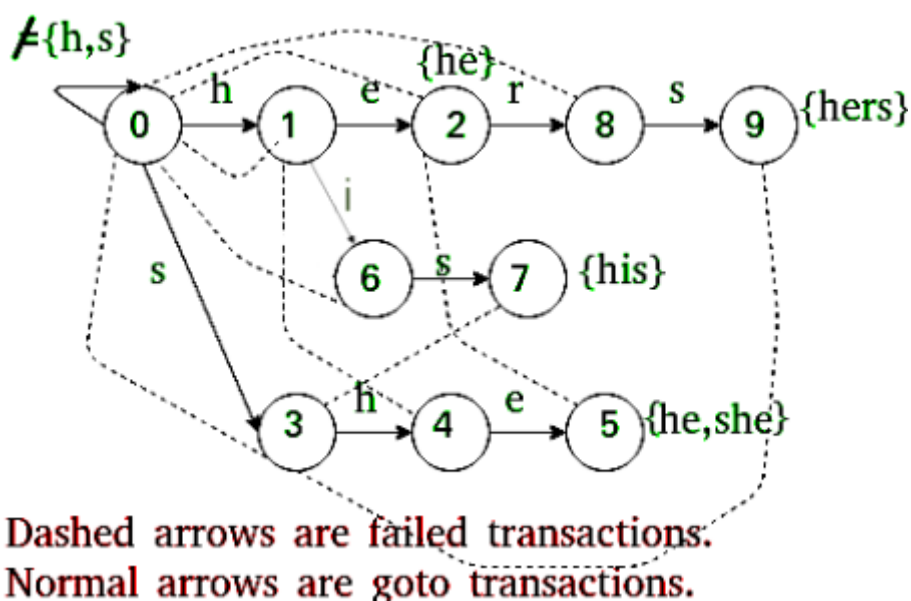
Trie for Arr[] = { he , she, his , hers }



Gambar 6. trie untuk {he,she,his,hers}

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

Kemudian trie dibuat menjadi automaton dengan menambahkan failure links. Pada contoh diatas, 's' di "hers" akan kembali ke root, lalu 'e' di "he" akan kembali ke root, dan 'h' pada "she" akan kembali ke h di root.

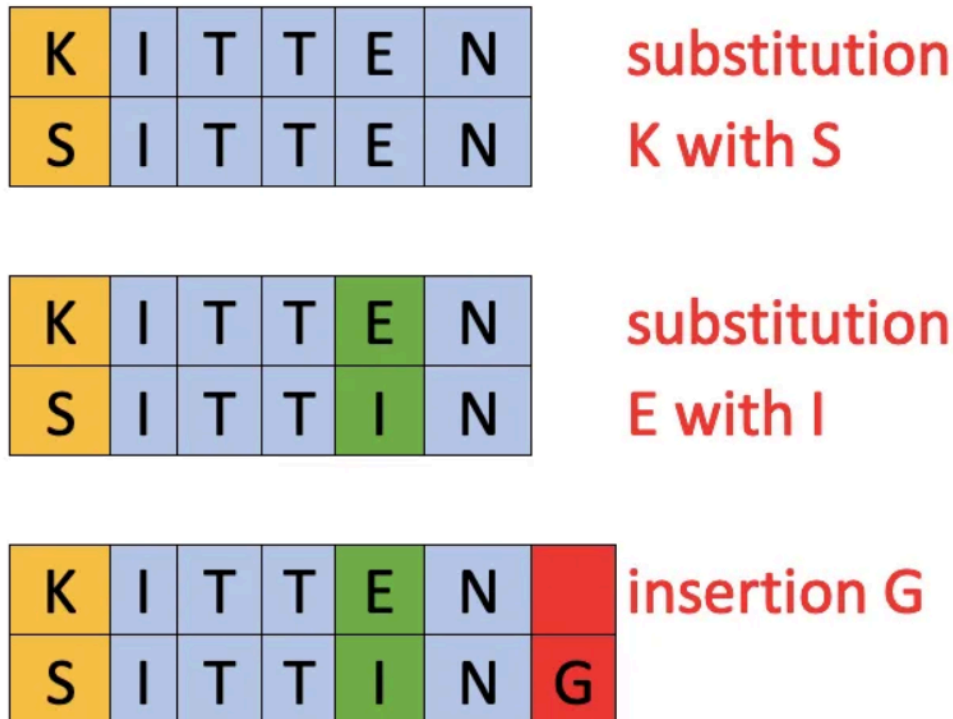


Gambar 7. trie yang sudah ditransformasi menjadi automaton

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

2.4 Levenshtein Distance

Algoritma Levenshtein Distance, atau disebut juga Algoritma Edit Distance, merupakan sebuah metode untuk mengukur kemiripan antara 2 string. Algoritma ini dikembangkan oleh Vladimir Levenshtein pada 1965. Algoritma ini mengkalkulasi banyak minimal dari perubahan satu karakter pada string yang diperlukan untuk mengubah suatu string menjadi string lainnya.



Gambar 6. Cara kerja Algoritma Levenshtein Distance

(sumber: Tejaswi, Y. (n.d.). *The Levenshtein distance algorithm: A string metric for measuring the difference between two sequences*. Medium

https://miro.medium.com/v2/resize:fit:1400/format:webp/1*4tNx1k945GctZrGGL1kBBQ.png)

Pada contoh kasus seperti di atas, dapat dilihat bagaimana cara algoritma ini bekerja. Kasus di atas adalah pengubahan string1 = 'KITTEN' ke string2 = 'SITTING'. Program mengkalkulasikan perbedaan dari kedua string tersebut, pertama adalah substitusi karakter 'K' dengan S, 'E' dengan 'I', dan terakhir adalah memasukkan 'G' ke string 1. Dengan Demikian, diperlukan 3 kali pengubahan karakter.

2.5. Aplikasi CV Analyzer

Aplikasi CV Analyzer adalah sebuah aplikasi yang dapat mencari sebuah *keyword* spesifik dari cv-cv yang di upload ke aplikasi. Dengan menggunakan aplikasi ini, pengguna dapat dengan efisien mencari kualitas-kualitas dari berbagai pelamar hanya dengan memasukan *keyword*.

Program bekerja dengan pertama melakukan regex matching pada cv yang baru diupload untuk mencari bagian bagian cv yang umum digunakan seperti *summary*, *skills*, dan *education*. Informasi-informasi ini disimpan pada sebuah database MySql untuk digunakan kembali. Kemudian, pengguna dapat melakukan *query* untuk kata-kata yang ingin dicari pada berbagai cv yang telah di masukan ke aplikasi. Pencarian pertama menggunakan *exact matching* dengan menggunakan algoritma Knuth-Morris-Pratt, Boyer-Moore, atau Aho-Corasick. Ketika *Exact Matching* tidak berhasil, program mencoba melakukan *fuzzy matching* dengan menggunakan Levenshtein distance. Hasil query adalah data-data yang diambil dari cv diawal dan cv nya itu sendiri.

BAB 3

ANALISIS PEMECAHAN MASALAH

3.1. Langkah-Langkah Pemecahan Masalah

Langkah-langkah pemecahan masalah ini dimulai dengan mengekstrak teks dari dokumen CV sehingga berbentuk sebuah string panjang agar dapat melakukan langkah-langkah selanjutnya. Lalu, pengguna akan memasukkan kata kunci yang kemudian dicocokkan dengan teks CV dengan pendekatan *pattern matching* dengan algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick (AC), serta didukung oleh *fuzzy algorithm* seperti algoritma Levenshtein Distance untuk mengatasi *typo*. Informasi-informasi penting seperti nama, email, keahlian, pengalaman kerja, dan pendidikan diekstrak menggunakan *regex*. Hasil ekstraksi tersebut selanjutnya dimasukkan ke dalam database berbasis MySQL. Sistem akan mengeluarkan daftar CV yang memiliki kecocokan kata kunci terbanyak dan memberikan opsi untuk melihat detail ringkasan dan file CV asli.

3.2. Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma KMP dan BM.

Ketika sudah didapatkan teks dan patternnya, program akan membuat elemen-elemen sebagai berikut:

1. Failure Function atau Border Function - Knuth-Morris-Pratt

Failure Function adalah sebuah tabel yang berisi indeks pada *pattern* dan panjang dari prefix yang ada di suffix juga jika pencocokan berakhir pada indeks tersebut. Contoh pada pembuatan Failure Function untuk “abaaba”, pada indeks 2 (0-indexing), akan ada sub-string yang berada di prefix dan juga suffix, yaitu ‘a’. Untuk indeks 5, akan ada ‘aba’ yang memiliki panjang 3. Berikut tabel lengkapnya:

Indeks	0	1	2	3	4
B(i)	0	0	1	1	2

2. L Function - Boyer-Moore

L function merupakan sebuah map yang menunjukkan kemunculan terakhir dari sebuah char yang ada di alfabet yang digunakan. Contoh pada kata “ambatunat”, akan terbuat tabel:

Char	a	m	b	t	u	n	lainnya
L()	7	1	2	8	5	6	-1

Map ini akan digunakan untuk mempercepat perhitungan pergeseran untuk algoritma.

3.3. Fitur fungsional dan Arsitektur Aplikasi GUI

3.3.1. Fitur Fungsional

ID	Fitur Fungsional	Penjelasan
F01	Pencarian Kandidat Berbasis Kata Kunci	Pengguna dapat mencari kandidat dengan memasukkan satu atau lebih kata kunci yang dipisahkan koma, memilih algoritma pencarian (KMP, BM, atau AC), dan membatasi jumlah hasil teratas yang ditampilkan.
F02	Pencocokan Ganda (Exact & Fuzzy Match)	Sistem melakukan pencarian dalam dua tahap: pertama, <i>exact match</i> menggunakan algoritma pilihan (KMP, BM, atau AC), lalu dilanjutkan dengan <i>fuzzy match</i> menggunakan Levenshtein Distance untuk kata kunci yang tidak ditemukan, guna mengatasi salah ketik.
F03	Tampilan Hasil Pencarian Interaktif	Hasil pencarian ditampilkan dalam bentuk CVCard yang memuat nama, total kecocokan, dan rincian kata kunci yang ditemukan. Aplikasi juga menampilkan waktu eksekusi kedua pencarian (<i>Exact</i> dan <i>Fuzzy</i>).
F04	Ekstraksi dan Rangkuman CV Otomatis	Sistem dapat mengekstrak teks dari file PDF, lalu mem-parsing-nya menggunakan Regex untuk mendapatkan informasi terstruktur seperti ringkasan, keahlian, riwayat kerja, dan pendidikan.
F05	Akses Langsung ke Dokumen CV	Setiap kartu hasil pencarian menyediakan tombol "Summary" untuk melihat rangkuman hasil ekstraksi dan tombol "View CV" untuk membuka dokumen PDF asli milik kandidat.

3.3.2. Arsitektur Aplikasi GUI

Aplikasi GUI yang kami bangun menggunakan framework **Flet**. Secara konseptual, arsitektur aplikasi dapat dipecah menjadi tiga layeyang saling berinteraksi: Presentation Layer, Application Logic Layer, dan Data Processing Layer.

1. Presentation Layer

Lapisan untuk segala hal yang berkaitan dengan GUI. Dibangun menggunakan framework Flet, lapisan ini berfungsi untuk menampilkan data kepada pengguna dan menangkap interaksi dari mereka.

Tampilan Utama : Terdapat dua tampilan utama:

Halaman Pencarian: Antarmuka awal tempat pengguna berinteraksi untuk memasukkan kriteria pencarian.

Halaman Rangkuman: Tampilan detail yang menyajikan hasil ekstraksi informasi CV secara terstruktur.

CVCard: Untuk menampilkan hasil pencarian secara konsisten dan informatif, dibuat komponen CVCard yang dapat digunakan kembali. Komponen ini merangkum data seorang kandidat ke dalam sebuah kartu visual.

2. Application Logic Layer

Lapisan ini bertindak sebagai "otak" dari aplikasi. Kelas ATSTApp dalam app_gui.py menjadi komponen sentral yang berfungsi sebagai controller.

State Management: Kelas ATSTApp mengelola seluruh keadaan aplikasi, seperti kata kunci yang dimasukkan, algoritma yang dipilih, dan daftar hasil pencarian yang akan ditampilkan.

Event Handling: Fungsi-fungsi di dalam kelas menangani semua input dan aksi dari pengguna yang diterima oleh Presentation Layer.

Orkestrasi Alur Kerja: Lapisan ini menjembatani komunikasi antara antarmuka dan pemrosesan data. Saat pengguna menekan tombol "Search", lapisan inilah yang akan memanggil fungsi-fungsi yang relevan di Data Processing Layer dan kemudian mengirimkan hasilnya kembali ke Presentation Layer untuk ditampilkan.

3. Data Processing Layer

Lapisan ini terisolasi dari antarmuka pengguna.

Modul Ekstraksi Data:

pdf_extractor.py: Menggunakan library PyPDF2 untuk mengekstraksi teks mentah dari dokumen PDF.

cv_extractor.py: Menggunakan Regular Expressions (Regex) untuk mem-parsing teks mentah menjadi data terstruktur (misalnya, keahlian, pengalaman kerja).

Modul Algoritma: Direktori src/algorithm/ berisi implementasi untuk setiap algoritma pattern matching dan fuzzy matching.

3.4. Ilustrasi Kasus

3.4.1. Ilustrasi Kasus dengan Algoritma KMP

Pencarian kata "jawa" pada teks "menurut pendapat saya, kunci kemenangan adalah jawa. siapa yang bisa menguasai jawa, itulah yang menang." Pertama dibuat LPS atau Border

Function:

indeks	0	1	2
B(i)	0	0	0

karena hasil border function semuanya 0, pencarian berjalan seperti bruteforce dimana semua karakter di cek satu-persatu. Didapatkan jawa pada posisi 47 dan 79.

3.4.2. Ilustrasi Kasus dengan Algoritma BM

Pencarian kata “jawa” pada teks “menurut pendapat saya, kunci kemenangan adalah jawa. siapa yang bisa menguasai jawa, itulah yang menang.”

Pertama dibuat L() Function sebagai berikut:

char	j	a	w	lainnya
L()	0	3	2	-1

Program pertama mencocokkan “menu” dengan “jawa” dimana $u \neq a$ sehingga digeser $\text{len}(\text{jawa}) - (-1) = 5$ indeks. Indeks yang baru adalah “rut “ dengan “jawa” dimana $u \neq a$ sehingga digeser kembali 5 indeks. Hal tersebut di lanjut hingga mencapai “ai j” dimana akan dimatch ‘j’ dengan ‘a’. Karena ‘j’ terdapat di dalam L() function maka digeser sebesar $\text{len}(\text{jawa}) - (0) = 4$. Alhasil didapatkan satu match “jawa”.

Algoritma melanjutkan ke indeks selanjutnya dengan melakukan shift sebanyak $\text{len}(\text{jawa}) + 1 = 5$ kemudian mengulang langkah algoritma Boyer-Moore. Didapatkan kembali pada “ jaw” pada jawa ke dua. Karena ‘w’ terdapat dalam L() function, dilakukan shift $\text{len}(\text{jawa}) - (2) = 2$ mendapatkan kata jawa kedua.

3.4.3. Ilustrasi Kasus dengan Algoritma AC

Pencarian kumpulan kata kunci {"he", "she", "his", "hers"} pada teks “yashehishers”.

Pertama, program membangun sebuah Automaton yang menggabungkan semua kata kunci ke dalam satu struktur. Struktur ini terdiri dari Trie (keyword tree) yang ditambahkan dengan failure links. Trie memetakan alur karakter untuk setiap kata kunci, misalnya s -> h -> e untuk "she". Kemudian, failure links dibuat untuk menghubungkan state. Contohnya, state yang dicapai setelah membaca "she" akan memiliki failure link ke state untuk "he", karena "he" merupakan sufiks dari "she" yang juga merupakan sebuah kata kunci. Hal ini memungkinkan pencarian untuk tidak mengulang dari awal ketika terjadi ketidakcocokan.

Program pertama memulai dari root dan membaca teks “yashehishers”. Karakter y dan a tidak memiliki transisi, sehingga dilewati. Saat mencapai s -> h -> e, program berada di state akhir untuk "she". Karena state ini juga terhubung dengan "he" melalui failure link, maka program mendeteksi dua match sekaligus: "she" yang berakhir di indeks 4 dan "he" yang berakhir di indeks 4.

Selanjutnya, program membaca karakter h. Dari state "she", tidak ada transisi untuk h, sehingga program mengikuti failure link ke state "he", lalu ke root. Dari root, program

membaca h dan berpindah ke state "h". Proses dilanjutkan dengan i -> s, mencapai state akhir untuk "his". Ditemukan satu match "his" yang berakhir pada indeks 7. Algoritma terus berjalan, memanfaatkan failure links setiap kali terjadi mismatch hingga seluruh teks selesai dibaca. Alhasil, dengan hanya satu kali pembacaan teks, semua kemunculan dari semua kata kunci dapat ditemukan secara efisien.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1. Spesifikasi Teknis Program

Program dibagi menjadi beberapa file python dengan pencampuran paradigma prosedural dan orientasi objek. Berikut file-file yang ada pada proyek kali ini.

1. `boyer_moore.py`

Memiliki fungsi `compute_l_function` dan `bm_search`. Memiliki dependency terhadap `pdf_extractor.py`.

- `compute_l_function(string) -> dict[str,int]`
Menerima parameter pattern dan akan menghasilkan l function dari pattern tersebut. Memiliki kompleksitas $O(m)$. Function ini berguna untuk dipakai pada function `bm_search`.
- `bm_search(string, string) -> List[int]`
Menerima pattern dan teks lengkap dalam bentuk string. Program akan menggunakan algoritma Bayer-Moore untuk mengkalkulasi indeks ditemukannya pattern pada string teks. Memiliki kompleksitas $O(m+n)$ dimana $O(m)$ dari `compute_l_function`.

2. `pdf_extractor.py`

Memiliki fungsi `extract_text_pypdf2` untuk melakukan pengambilan data teks String dari sebuah file pdf. Memiliki dependency terhadap library PyPDF2.

- `extract_text_pypdf2(String) -> String`
Menerima parameter berupa path ke lokasi pdf. Path dapat berupa absolute atau relative tapi disarankan menggunakan relative untuk menghindari hijinks dari perbedaan OS. Function akan mengembalikan hasil teks yang berhasil di ekstrak dari file pdf yang berada pada path tersebut.

3. `aho_corasick.py`

- `class TrieNode`
Mewakili node dalam struktur Trie
- `class AhoCorasick`
 1. `init`
Membuat struktur Trie dari daftar kunci, lalu membentuk *failure links*
 2. `build_trie`
Menyisipkan semua kata kunci ke dalam Trie secara karakter per karakter
 3. `build_failure_links`
Menggunakan BFS untuk menautkan *failure link* antar node
 4. `search(text)`
Menjalankan pencarian semua kata kunci dalam teks,

4. `KMP.py`

Memiliki fungsi `compute_lps_array` dan `kmp_search`.

- `compute_lps_array`
Menerima parameter pattern, menghitung *Longest Proper Prefix* dari string pattern. Memiliki kompleksitas $O(m)$ dengan m adalah panjang pattern.

- `kmp_search`
Menerima parameter text dan pattern, menjalankan pencarian pattern dalam text menggunakan lps array. Memiliki kompleksitas $O(n+m)$ dengan m adalah pembuatan LPS array dan m adalah pencarian dalam text menggunakan LPS.
5. `levenshtein.py`
Memiliki fungsi `levenshtein_distance` dan `levenshtein_search`.
 - `levenshtein_distance`
Menerima parameter pattern dan substring, menghitung jarak edit minimum untuk mengubah substring menjadi pattern. Memiliki kompleksitas $O(mn)$ dengan m adalah panjang substrirng dan n panjang pattern.
 - `levenshtein_search`
Menerima parameter pattern, text, dan threshold. Mencari semua kemunculan pattern dalam text dengan toleransi kesalahan \leq threshold. memiliki kompleksitas $O(k*m^2)$ dengan k panjang text dan m panjang pattern.
 6. `app_gui.py`
Aplikasi berbasis framework Flet yang digunakan untuk mencari dan menganalisis CV dengan berbagai algoritma pencocokan. Menampilkan antarmuka pencarian, detail pelamar, ringkasan, serta membuka file secara langsung.
 7. `cv_extractor.py`
Melakukan ekstraksi informasi utama dari CV, seperti *summary*, *skills*, *experience*, dan *education*. Setiap bagian informasi diolah secara terpisah menggunakan *regex* untuk mengenali pola umum dalam CV.
 8. `database.py`
Sistem manajemen basis data berbasis MySQL, yang mencakup koneksi database, pembuatan dan pengisian database, serta fitur-fitur operasional lainnya.

4.2. Tata Cara Penggunaan Program

1. Clone Repository


```
``bash
git clone https://github.com/Starath/Tubes3\_WokdeTok.git
cd Tubes3_WokdeTok
``
```
2. Install Dependencies


```
``bash
pip install -r requirements.txt
``
```
3. Masuk Ke MySQL
 - ubuntu

```
``bash
sudo mysql -u root -p
``
```

- windows

```

'''bash
cd \path\to\your\MySQL\bin
mysql -u root -p
'''

```

4. Jalankan Command Berikut

```
'''
```

```

CREATE USER 'app_user'@'localhost' IDENTIFIED BY 'bitchass';
GRANT ALL PRIVILEGES ON *.* TO 'app_user'@'localhost';
FLUSH PRIVILEGES;
SELECT user, host FROM mysql.user WHERE user = 'app_user';
EXIT;
'''

```

5. Jalankan program GUI

6. Masukan keyword yang diinginkan pada GUI

4.3. Hasil Pengujian

4.3.1. Test Case 1

Dicari kalimat “and now i am become death” dari script oppenheimer : https://assets.scripslug.com/live/pdf/scripts/oppenheimer-2023.pdf	
Algoritma	Hasil
Knuth-Morris-Pratt	<pre> Time taken: 0.062267 seconds Text: 222860 Char long Pattern: 'and now i am become death' LPS Array for 'and now i am become death': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0] Pattern found at indices: [30270] </pre>
Boyer-Moore	<pre> Time taken: 0.013907 seconds Text: 222860 Char long Pattern: 'and now i am become death' L Function Array for 'and now i am become death': {'a': 22, 'n': 4, 'd': 20, ' ': 19, 'o': 16, 'w': 6, 'l': 8, 'm': 17, 'b': 13, 'e': 21, 'c': 15, 't': 23, 'h': 24} Pattern found at indices: [30270] </pre>
Aho-Cosarick	<pre> Text: 222860, Keywords: ['and now i am become death'] time : 0.042384 seconds Result: {'and now i am become death': [30270]} </pre>

4.3.2. Test Case 2

Dicari kata-kata {honey, bee, barry, ken} dari script bee movie 2007: https://gist.github.com/MattIPv4/045239bc27b16b2bcf7a3a9a4648c08a	
Algoritma	Hasil

<h1>Knuth-Morris-Pratt</h1>	<pre> Time taken: 0.068169 seconds Pattern: 'ken' L Function Array for 'ken': [0, 0, 0] Pattern found at indices: [14284, 14330, 16232, 26693, 34411, 34718, 41762, 42352, 42846, 43589, 43874, 44495, 44777, 49075, 55137, 67157] ----- Pattern: 'harry' L Function Array for 'harry': [0, 0, 0, 0, 0] Pattern found at indices: [2945, 3250, 3747, 7716, 7964, 8108, 8380, 12635, 13782, 17869, 23636, 26539, 27688, 27646, 27692, 27729, 28086, 28943, 29113, 30439, 30502, 30951, 30966, 312 04, 32131, 32995, 34432, 36057, 36326, 37014, 38256, 38688, 38918, 39014, 39475, 43954, 46992, 50271, 50748, 51084, 51534, 52117, 52515, 52553, 52652, 52816, 53163, 53244, 53678, 54190, 55807, 61268, 65407, 66234] ----- Pattern: 'honey' L Function Array for 'honey': [0, 0, 0, 0, 0, 0] Pattern found at indices: [2945, 3250, 3747, 7716, 7964, 8108, 8380, 12635, 13782, 17869, 23636, 26539, 27688, 27646, 27692, 27729, 28086, 28943, 29113, 30439, 30502, 30951, 30966, 312 04, 32131, 32995, 34432, 36057, 36326, 37014, 38256, 38688, 38918, 39014, 39475, 43954, 46992, 50271, 50748, 51084, 51534, 52117, 52515, 52553, 52652, 52816, 53163, 53244, 53678, 54190, 55807, 61268, 65407, 66234] ----- Pattern: 'bee' L Function Array for 'bee': [0, 0, 0] Pattern found at indices: [83, 223, 272, 2114, 2186, 2814, 3468, 3495, 3888, 3929, 4291, 4835, 5971, 7081, 8753, 9497, 9770, 10835, 11047, 11224, 13256, 13788, 13853, 14262, 15387, 165 88, 17271, 17568, 17688, 17627, 17936, 19399, 20127, 21549, 21887, 21929, 23823, 23856, 23874, 23902, 23110, 23149, 23164, 23164, 23229, 23360, 23578, 24164, 25081, 26051, 26330, 26593 27455, 28057, 28754, 28902, 29131, 29778, 30043, 30945, 31412, 31649, 31920, 32298, 32506, 32599, 32688, 33088, 33412, 33434, 33494, 33512, 33586, 33793, 34606, 34699, 35056, 35143, 35307, 35657, 36331, 36791, 36988, 37223, 37870, 38249, 38328, 39068, 39102, 39218, 39324, 39345, 39615, 40031, 40181, 40335, 41001, 42682, 44039, 44092, 44363, 45900, 45412, 45540, 45852, 45 973, 46142, 46195, 46497, 46788, 48303, 48336, 48340, 48397, 48849, 49087, 49965, 50013, 50041, 50449, 50484, 50842, 50983, 50524, 50545, 50599, 50793, 51247, 51269, 51480, 51480, 51490, 51496, 51494, 51515, 54314, 55588, 56349, 56366, 57724, 58797, 59356, 59905, 60679, 60887, 61299, 61564, 61620, 61825, 61830, 61838, 61843, 61971, 63353, 63380, 63398, 63416, 63434, 63452, 63607, 63605, 63783, 63888, 63826, 63844, 63945, 64356, 65680, 65675, 65960, 66207, 67101, 67434] </pre>
<h1>0.068169 seconds</h1>	
<h1>Boyer-Moore</h1>	<pre> Time taken: 0.032726 seconds Pattern: 'ken' L Function Array for 'ken': [0, 0, 0, 0, 0, 0] Pattern found at indices: [14284, 14330, 16232, 26693, 34411, 34718, 41762, 42352, 42846, 43589, 43874, 44495, 44777, 49075, 55137, 67157] ----- Pattern: 'harry' L Function Array for 'harry': [0, 0, 0, 0, 0, 0] Pattern found at indices: [484, 564, 1080, 1165, 1520, 6867, 7973, 9903, 9954, 9963, 19159, 20522, 22359, 23645, 23743, 26581, 31138, 31556, 31767, 32912, 33302, 36406, 36980, 41978, 4 2359, 42863, 44983, 46061, 50892, 51870, 54846, 55522, 55758, 58394, 58973, 59776, 60990, 60290, 60359, 61851, 63360, 64456, 64657, 64955, 65751, 65938, 66479, 66789, 66972, 67091] ----- Pattern: 'honey' L Function Array for 'honey': [0, 0, 0, 0, 0, 0, 0, 0] Pattern found at indices: [2945, 3260, 3747, 7716, 7964, 8108, 8380, 12635, 13782, 17869, 23636, 26539, 27688, 27646, 27692, 27729, 28086, 28943, 29113, 30439, 30502, 30951, 30966, 312 04, 32131, 32995, 34432, 36057, 36326, 37014, 38256, 38688, 38918, 39014, 39475, 43954, 46992, 50271, 50748, 51084, 51534, 52117, 52515, 52553, 52652, 52816, 53163, 53244, 53678, 54190, 55807, 61268, 65407, 66234] ----- Pattern: 'bee' L Function Array for 'bee': [0, 0, 0, 0, 0, 0] Pattern found at indices: [83, 223, 272, 2114, 2186, 2814, 3468, 3495, 3888, 3929, 4291, 4835, 5971, 7081, 8753, 9497, 9770, 10835, 11047, 11224, 13256, 13788, 13853, 14262, 15387, 165 88, 17271, 17568, 17688, 17627, 17936, 19399, 20127, 21549, 21887, 21929, 23823, 23856, 23874, 23902, 23110, 23149, 23164, 23164, 23229, 23360, 23578, 24164, 25081, 26051, 26330, 26593 27455, 28057, 28754, 28902, 29131, 29778, 30043, 30945, 31412, 31649, 31920, 32298, 32506, 32599, 32688, 33088, 33412, 33434, 33494, 33512, 33586, 33793, 34606, 34699, 35056, 35143, 35307, 35657, 36331, 36791, 36988, 37223, 37870, 38249, 38328, 39068, 39102, 39218, 39324, 39345, 39615, 40031, 40181, </pre>

4.3.3. Test Case 3

CV Digital Media 10005171 Mencari {bronze, silver, gold,supervise,master}	
Algoritma	Hasil
Knuth-Morris-Pratt	<pre> Time taken: 0.000610 seconds Pattern: 'bronze' L Function Array for 'bronze': [0, 0, 0, 0, 0, 0] Pattern found at indices: [893, 1468] Pattern: 'silver' L Function Array for 'silver': [0, 0, 0, 0, 0, 0] Pattern found at indices: [1074, 1401] Pattern: 'gold' L Function Array for 'gold': [0, 0, 0, 0] Pattern found at indices: [1306] Pattern: 'master' L Function Array for 'master': [0, 0, 0, 0, 0, 0] Pattern found at indices: [4585] Pattern: 'supervise' L Function Array for 'supervise': [0, 0, 0, 0, 0, 0, 0, 1, 0] Pattern found at indices: [2616] </pre>
Boyer-Moore	<pre> Time taken: 0.003728 seconds Pattern: 'bronze' L Function Array for 'bronze': {'b': 0, 'r': 1, 'o': 2, 'n': 3, 'z': 4, 'e': 5} Pattern found at indices: [893, 1468] Pattern: 'silver' L Function Array for 'silver': {'s': 0, 'i': 1, 'l': 2, 'v': 3, 'e': 4, 'r': 5} Pattern found at indices: [1074, 1401] Pattern: 'gold' L Function Array for 'gold': {'g': 0, 'o': 1, 'l': 2, 'd': 3} Pattern found at indices: [1306] Pattern: 'master' L Function Array for 'master': {'m': 0, 'a': 1, 's': 2, 't': 3, 'e': 4, 'r': 5} Pattern found at indices: [4585] Pattern: 'supervise' L Function Array for 'supervise': {'s': 7, 'u': 1, 'p': 2, 'e': 8, 'r': 4, 'v': 5, 'i': 6} Pattern found at indices: [2616] </pre>
Aho-Cosarick	<pre> Text: 5699, Keywords: ['bronze', 'silver', 'gold', 'master', 'supervise'] time : 0.000847 seconds Result: {'bronze': [893, 1468], 'silver': [1074, 1401], 'gold': [1306], 'supervise': [2616], 'master': [4585]} </pre>

BAB 5

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dari tugas besar 3 IF2211 Strategi algoritma ini, kami telah berhasil membuat aplikasi GUI yang bisa digunakan untuk memanfaatkan *pattern matching* untuk Membangun Sistem ATS (Applicant Tracking System) Berbasis CV Digital. Untuk menyelesaikan tugas ini, kami menggunakan bahasa pemrograman Python dengan framework GUI yaitu Flet. Didapatkan bahwa algoritma Boyer-Moore lebih cepat daripada KMP karena kata-kata yang dipakai untuk pencarian cenderung tidak memiliki pengulangan prefix-suffix sehingga algoritma KMP lebih seperti algoritma Bruteforce. Kami juga sangat terkejut bahwa python dapat melakukan algoritma Boyer-Moore dengan waktu 0.5s untuk 1 juta karakter.

5.2. Saran

Waktu diperpanjang, jangan mengerjakan dekat dengan deadline, dan gunakan sqlite next time.

LAMPIRAN

Repository: https://github.com/Starath/Tubes3_WokdeTok

Link Video:

Tabel Progress:

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

REFERENSI

- [1]. R. Munir, "Pencocokan string (string matching) dengan algoritma brute force, KMP, Boyer-Moore" Institut Teknologi Bandung, 2025. [Online]. Available: [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). [Diakses: Juni. 14, 2025].
- [2]. R. Munir, "Pencocokan string dengan regular expression (regex)" Institut Teknologi Bandung, 2025. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf). [Diakses: Juni. 14, 2025].
- [3]. UNSW Sydney, "Lecture 9a: String Algorithms," School of Computer Science and Engineering, COMP2521 Data Structures and Algorithms, 2019. [Online]. Available: <https://cgi.cse.unsw.edu.au/~cs2521/19T1/lects/week09a/#s14>. [Diakses: Juni. 14, 2025].
- [4]. Tejaswi, Y. (n.d.). *The Levenshtein distance algorithm: A string metric for measuring the difference between two sequences.* Medium. <https://medium.com/@tejaswiyadav221/the-levenshtein-distance-algorithm-a-string-metric-for-measuring-the-difference-between-two-269afbbddd34> [Diakses: Juni. 14, 2025]

