

LAPORAN TUGAS BESAR 3

IF2211 STRATEGI ALGORITMA

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



Disusun Oleh:

“ikan dan pisang”

Bertha Soliany Frandi	13523026
Rafen Max Alessandro	13523031
Grace Evelyn Simon	13523087

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

BAB 1	4
DESKRIPSI TUGAS	4
BAB 2	6
LANDASAN TEORI	6
2.1 Penjelasan Algoritma	6
2.1.1 Algoritma Knuth-Morris-Pratt (KMP)	6
2.1.2 Algoritma Boyer-Moore (BM)	8
2.1.3 Algoritma Aho-Corasick (AC)	10
2.1.4 Algoritma Levenshtein Distance	13
2.1.5 Regular Expressions (Regex)	16
2.3 Penjelasan Aplikasi Desktop	19
BAB 3	20
ANALISIS PEMECAHAN MASALAH	20
3.1 Langkah-langkah Pemecahan Masalah	20
3.2 Proses Pemetaan Masalah	22
3.3.1 Fitur Fungsional Aplikasi Desktop	23
3.3.2 Arsitektur Aplikasi Desktop	24
3.4 Contoh Ilustrasi Kasus	25
BAB 4	27
IMPLEMENTASI DAN PENGUJIAN	27
4.1 Struktur Data	27
4.2 Fungsi dan Prosedur Program	28
4.3 Tata Cara Penggunaan Program	71
4.4 Hasil Pengujian	73
4.4.1 Pengujian Biasa	73
4.4.2 Pengujian Lainnya	81
4.5 Analisis Hasil Pengujian	84
BAB 5	86
KESIMPULAN, SARAN, DAN REFLEKSI	86
5.1 Kesimpulan	86
5.2 Saran	87
5.3 Refleksi	87
LAMPIRAN	88
Tautan Repository Github	88
Tautan Video	88
Hasil Akhir Tugas Besar	88
DAFTAR PUSTAKA	89

DAFTAR GAMBAR

Gambar 1. CV ATS dalam Dunia Kerja	5
(Sumber: https://www.antaranews.com/)	5
Gambar 2. Contoh Penerapan Algoritma KMP	8
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)	8
Gambar 3. Contoh Penerapan Algoritma KMP	9
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)	9
Gambar 4. Contoh prefix tree untuk pattern ['Java', 'Rad', 'Rand', 'Rau', 'Raum', dan 'Rose']	10
(Sumber: cp-algorithms.com/string/aho_corasick.html)	11
Gambar 5. Contoh failure links dan output links untuk prefix tree	11
(Sumber: https://www.youtube.com/watch?v=XWujo7KQL54)	11
Gambar 6. Matriks representasi nilai Levensthein Distance untuk setiap sub-string	14
(Sumber: https://www.youtube.com/watch?v=XYi2-LPrwm4)	14
Gambar 7. Inisialisasi nilai untuk baris terakhir dan kolom terakhir	15
(Sumber: https://www.youtube.com/watch?v=XYi2-LPrwm4)	15
Gambar 8. Tahapan rekursivitas untuk mendapatkan Levensthein Distance antar string.	16
(Sumber: https://www.youtube.com/watch?v=XYi2-LPrwm4)	16
Gambar 9. Kumpulan metakarakter dalam sintaks pencocokan menggunakan Regex	17
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf)	17
Gambar 10. Skema Basis Data CV ATS	19
(Sumber: https://docs.google.com/document/d/1aVJtQ9oj-WE7GrQwaHv3xzYcAMIsplSBLAX4SLBcPZQA/edit?tab=t.0)	19

BAB 1

DESKRIPSI TUGAS

Pada era digital dengan keamanan data dan akses yang 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). 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. Selain itu, sistem ini akan dihubungkan dengan identitas kandidat melalui basis data sehingga dapat terbentuk sebuah sistem yang dapat mengenali profil pelamar secara lengkap hanya dengan menggunakan CV digital.

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. 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. Dalam skema basis data, 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.

Secara umum, program yang dibuat memiliki langkah penggunaan sebagai berikut:

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



Gambar 1. CV ATS dalam Dunia Kerja

(Sumber: <https://www.antaranews.com/>)

BAB 2

LANDASAN TEORI

2.1 Penjelasan Algoritma

2.1.1 Algoritma Knuth-Morris-Pratt (KMP)

Algoritma *Knuth-Morris-Pratt* (KMP) adalah sebuah metode pencocokan string yang sangat efisien, dinamai sesuai dengan para pengembangnya, yakni Donald Knuth, James H. Morris, dan Vaughan Pratt. Tujuan utama dari algoritma KMP adalah melakukan proses pencocokan pola P dalam teks T dengan memindai dari kiri ke kanan, serupa dengan pendekatan *Brute Force*. Namun, perbedaan fundamental terletak pada cara KMP menangani ketidakcocokan (*mismatch*). KMP menggeser pola P secara lebih cerdas dengan memanfaatkan informasi yang terkandung dalam pola itu sendiri.

Algoritma KMP menggunakan fungsi pinggiran (*border function*) yang disebut juga dengan fungsi gagal (*failure function*) untuk menentukan seberapa jauh pola dapat digeser saat terjadi ketidaksesuaian. Fungsi ini memproses pola untuk menemukan kecocokan antara awalan (*prefix*) dan akhiran (*suffix*) pola itu sendiri. Sebuah *mismatch* terjadi ketika karakter pada posisi tertentu di teks *T* tidak sama dengan karakter pada posisi yang bersesuaian di pola *P*. Jika *i* adalah indeks pada *T* dan *j* adalah indeks pada *P*, maka *mismatch* terjadi jika *T*[*i*] = *P*[*j*]. Ketika *mismatch* terjadi, KMP akan menentukan seberapa jauh pola *P* dapat digeser ke kanan tanpa kehilangan potensi kecocokan yang mungkin ada.

Algoritma KMP terdiri dari dua fase utama, yakni tahap pra-pemrosesan (*border/failure function*) dan tahap pencarian.

1. Tahap Pra-pemrosesan

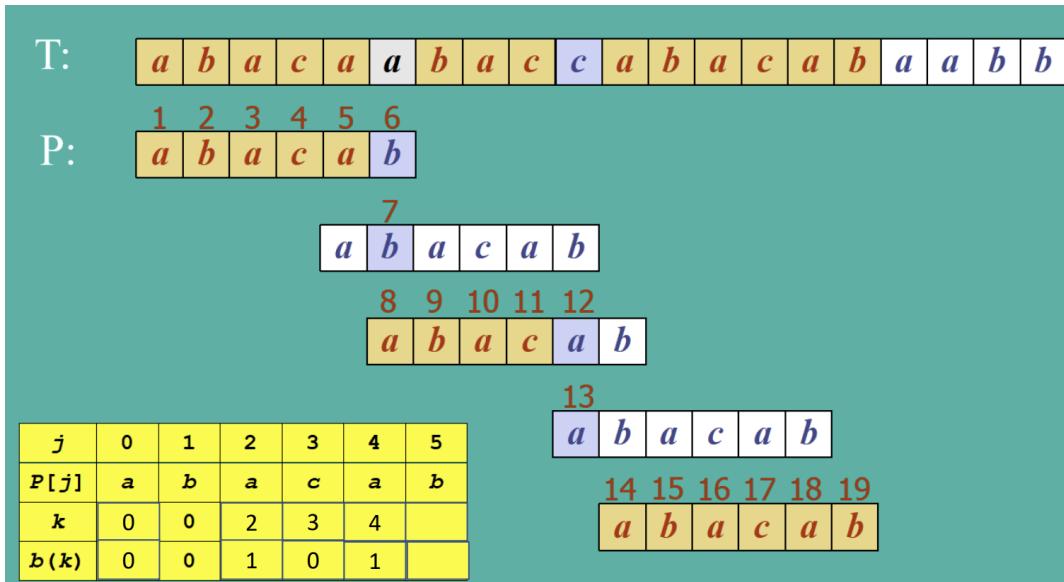
Fungsi ini, dilambangkan dengan *b*(*k*) atau *fail*(*k*), didefinisikan sebagai ukuran dari *prefix* terpanjang dari *substring* *P*[0..*k*] yang juga merupakan sebuah *suffix* dari substring *P*[1..*k*]. Tujuan dari fungsi pinggiran ini adalah untuk mengidentifikasi adanya tumpang tindih (*overlap*)

antara *prefix* dari pola dengan bagian lain dari pola itu sendiri. Informasi ini memungkinkan algoritma untuk menentukan pergeseran maksimum yang aman setelah terjadi *mismatch*, tanpa perlu membandingkan ulang karakter-karakter yang sudah pasti cocok berdasarkan struktur internal pola.

2. Tahap Pemrosesan

Setelah tabel fungsi pinggiran b dihitung, tahap pencarian dimulai. Algoritma KMP membandingkan karakter-karakter teks T dengan karakter-karakter pola P dari kiri ke kanan, menggunakan dua *pointer*: i untuk teks dan j untuk pola.

- Jika $T[i] = P[j]$, kedua pointer (i dan j) digeser ke kanan ($i++$, $j++$) untuk melanjutkan perbandingan.
- Jika j mencapai m (panjang pola), berarti seluruh pola telah cocok, dan lokasi $i - m$ dikembalikan sebagai tempat ditemukannya pola.
- Jika terjadi *mismatch*, yaitu $T[i] \neq P[j]$:
 - Jika $j > 0$, artinya ada beberapa karakter pola yang sudah cocok sebelumnya. Pada titik ini, KMP tidak langsung menggeser pola satu karakter ke kanan, melainkan menggunakan nilai dari fungsi pinggiran. *Pointer* j pada pola diubah menjadi $b[j - 1]$. Ini secara efektif menggeser pola ke kanan sedemikian rupa sehingga *prefix* dari P dengan panjang $b[j - 1]$ (yang diketahui juga merupakan *suffix* dari $P[0..j - 1]$) menjadi selaras dengan bagian teks yang baru saja cocok dengan *suffix* tersebut. *Pointer* i pada teks tidak diubah.
 - Jika $j = 0$ (*mismatch* terjadi pada karakter pertama pola), maka tidak ada informasi dari fungsi pinggiran yang bisa digunakan untuk pergeseran cerdas terkait *prefix*. Dalam kasus ini, pointer i pada teks digeser satu posisi ke kanan dan perbandingan dimulai lagi dari awal pola ($j = 0$).



Gambar 2. Contoh Penerapan Algoritma KMP

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

2.1.2 Algoritma Boyer-Moore (BM)

Algoritma Boyer-Moore (BM), dikembangkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977 adalah salah satu algoritma pencocokan string yang paling efisien dalam praktik. Prinsip dasar algoritma BM adalah melakukan pra-pemrosesan pada pola P untuk mengumpulkan informasi yang memungkinkan algoritma untuk melewati sebagian besar bagian teks T selama proses pencarian. Secara umum, algoritma BM cenderung berjalan lebih cepat seiring dengan bertambahnya panjang pola P .

Dua teknik fundamental yang menjadi ciri khas algoritma BM adalah teknik “*Looking-Glass*” yang melakukan perbandingan karakter antara pola dan teks dari kanan ke kiri, dimulai dari karakter terakhir pola dan teknik “*Character-Jump*” yang menggunakan informasi dari karakter di teks yang menyebabkan ketidakcocokan, serta informasi dari struktur pola untuk melakukan pergeseran (lompatan) pola.

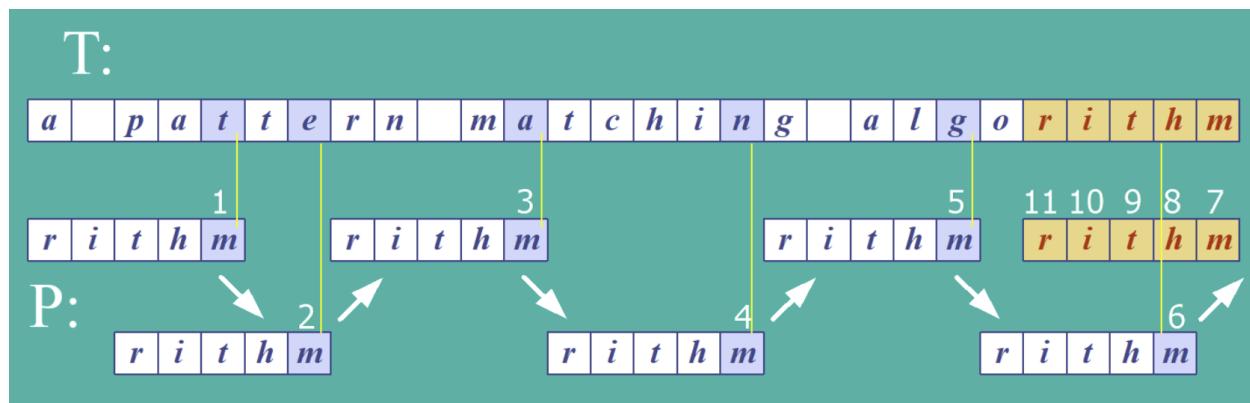
1. Teknik “*Looking-Glass*”

Teknik ini melibatkan proses pencocokan yang dimulai dari karakter terakhir pola ($P[m - 1]$) dan bergerak mundur menuju karakter pertama pola (P). Jika terjadi

ketidakcocokan pada karakter $P[j]$ dengan karakter teks yang bersesuaian, maka karakter-karakter pola $P[j + 1..m - 1]$ (jika ada) sudah dipastikan cocok dengan segmen teks yang relevan.

2. Teknik “Character-Jump” / Bad Character Heuristic

Ketika terjadi ketidakcocokan antara karakter $P[j]$ pada pola dan karakter $x = T[i]$ pada teks, heuristik ini menentukan seberapa jauh pola P dapat digeser ke kanan. Pergeseran ini didasarkan pada posisi kemunculan terakhir (paling kanan) dari karakter x di dalam pola P yang terletak di sebelah kiri dari posisi ketidakcocokan j saat ini. Untuk mengimplementasikan *Bad Character Heuristic* secara efisien, dilakukan pra-pemrosesan pada pola P untuk membuat sebuah tabel yang disebut fungsi kemunculan terakhir, $L(x)$. Fungsi $L(x)$ untuk setiap karakter x dalam alfabet didefinisikan sebagai indeks terbesar (paling kanan) k sedemikian sehingga $P[k] = x$. Jika karakter x tidak ada sama sekali dalam pola P , maka $L(x)$ biasanya diberi nilai -1. Tujuan dari fungsi $L(x)$ adalah untuk menyediakan informasi secara cepat mengenai seberapa jauh pola P dapat digeser sehingga karakter x di teks (yang menyebabkan *mismatch*) menjadi selaras dengan kemunculan terakhirnya di dalam pola.



Gambar 3. Contoh Penerapan Algoritma KMP

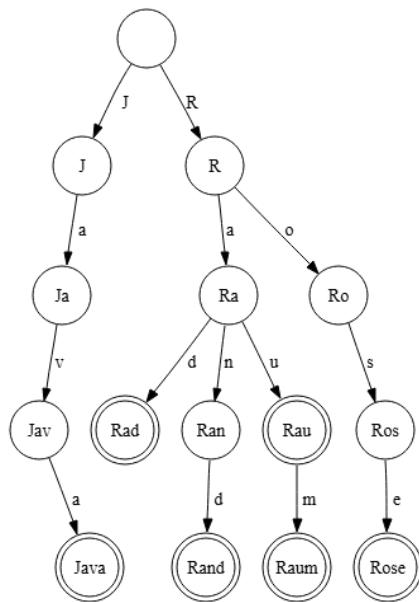
(Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf))

2.1.3 Algoritma Aho-Corasick (AC)

Algoritma Aho-Corasick merupakan algoritma yang diusulkan oleh Alfred Aho dan Margaret Corasick pada tahun 1975 sebagai algoritma pencarian beberapa pola dalam suatu teks dengan cepat. Algoritma ini didasarkan terhadap *finite state automaton* menggunakan dasar struktur data *trie* untuk digunakan dalam memproses teks. Penggunaan *automaton* memberikan keuntungan dibandingkan kedua algoritma sebelumnya yang hanya mencari satu pola dalam satu waktu tertentu. Algoritma ini bekerja dalam dua fase utama, yaitu *preprocessing* sebagai pembangunan automaton, dan *processing* sebagai pencocokan *pattern* dengan teks.

1. Preprocessing

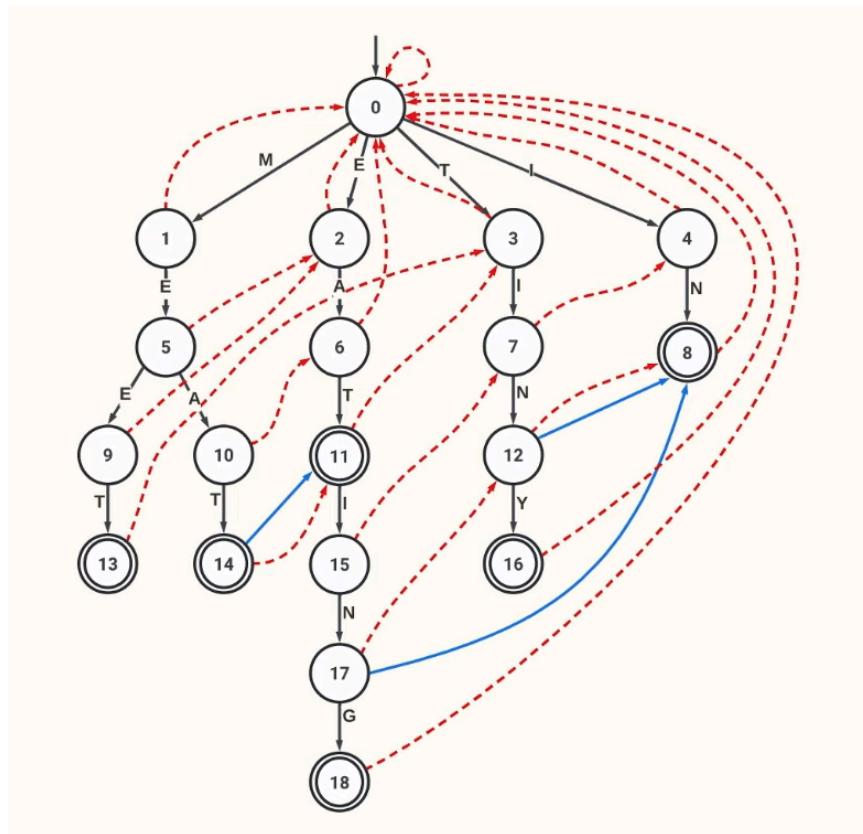
Pembangunan struktur data *automaton* merupakan pembangunan *trie* sebagai *prefix tree* yang memiliki atribut *failure links* dan *output links*. Semua *pattern* yang akan dicari dimasukkan ke dalam struktur *trie* dimana setiap simpul merepresentasikan *prefix* dari sebuah *pattern*. Karena memungkinkan bagi *pattern* untuk memiliki kesamaan *prefix* antara satu dengan yang lain, maka sebuah simpul dapat merepresentasikan *prefix* untuk satu atau lebih *pattern* yang akan dicocokkan. Hasil dari *prefix tree* adalah sebagai berikut.



Gambar 4. Contoh *prefix tree* untuk *pattern* ['Java', 'Rad', 'Rand', 'Rau', 'Raum', dan 'Rose']

(Sumber: cp-algorithms.com/string/aho_corasick.html)

Setelah membentuk *prefix tree*, dilakukan pembentukan terhadap *failure link*, yaitu simpul yang dituju jika huruf berikutnya pada teks bukan merupakan huruf yang membentuk salah satu *pattern* yang diharapkan. Contoh dari *failure link* adalah sebagai berikut.



Gambar 5. Contoh *failure links* dan *output links* untuk *prefix tree*

(Sumber: <https://www.youtube.com/watch?v=XWujo7KOL54>)

Setiap panah yang keluar dari sebuah simpul menunjukkan *failure link* yang dimiliki oleh simpul tersebut, dengan pembentukan *failure link* dimulai dari *root* hingga kedalaman terakhir untuk membentuk *links* yang saling terhubung satu dengan yang lain. Sebagai contoh, tahapan pembentukan *failure link* untuk simpul 9 adalah:

1. Sebuah simpul akan didatangi sesuai dengan huruf terakhir yang membentuk *prefix* tersebut, misalkan untuk simpul 9, simpul ini akan didatangi jika ditemukan huruf E setelah mendatangi simpul 5.
2. *Failure link* untuk simpul 9 kemudian diperiksa berdasarkan *failure link* dari simpul *parent*-nya, yaitu simpul 5. *Failure link* simpul 5 mengarah ke simpul 2, dan di simpul 2, tidak terdapat *child* yang didatangi oleh huruf yang sama untuk mendatangi oleh simpul 9, yaitu huruf E.
3. Maka, pengecekan *failure link* diteruskan lagi pada simpul 2, yang mengarah ke *root*. Di *root*, terdapat *child* yang didatangi oleh huruf E, yaitu simpul 2. Sehingga, *failure link* dari simpul 9 akan mengarah ke simpul 2.
4. Proses rekursif akan dilakukan bagi setiap simpul sesuai dengan prosedur tersebut.
5. Untuk memulai tahapan rekursif ini, *failure link* dari *root* diinisialisasi terlebih dahulu untuk mengarah ke dirinya sendiri, dan *failure link* dari simpul pada kedalaman pertama (*children* dari *root*) mengarah ke *root*. Dengan demikian, dapat dipastikan bahwa setiap *failure links* akan mengarah ke simpul yang lain, dengan kondisi basis ke *root* jika sama sekali tidak terdapat simpul huruf selanjutnya sebagai *children*.

Failure link mempercepat pencarian secara traversal dengan tidak mengulang dari awal setiap kali ditemukan kegagalan dalam melakukan pencocokan, baik untuk *pattern* maupun teks.

Komponen terakhir dari *automaton* adalah *output links* yang merepresentasikan bahwa sebuah *pattern* telah ditemukan, atau dengan kata lain menyatakan bahwa suatu simpul merupakan akhir dari salah satu *pattern*. Pada persoalan tugas besar, karena penggunaan *string matching* bertujuan untuk menyatakan jumlah *pattern* yang ditemukan dibandingkan posisi dari *pattern* dalam teks, maka *output links* digunakan untuk menambahkan jumlah *occurrences* dari *pattern* tersebut. Berbeda dengan *failure links*, *output links* tidak mengarahkan kemana pengecekan dilanjutkan, tetapi hanya

menyatakan *occurrences*, sehingga proses pengecekan tetap dilanjutkan ke *children* atau *failure links* dari simpul tersebut berdasarkan huruf selanjutnya pada teks.

2. Processing

Setelah *automaton* telah dibangun, algoritma akan memulai pemindaian teks per karakter untuk mencari semua *pattern* yang ingin dicari sekaligus. Iterasi dilakukan dengan mengikuti *trie* dari *root* dan melakukan transisi sesuai dengan huruf berikutnya pada teks, apakah menuju *failure links* atau *children* dari sebuah simpul, dan mencatat jumlah kemunculan *pattern* setiap kali ditemukan *output links* pada simpul.

2.1.4 Algoritma Levenshtein Distance

Algoritma *Levenshtein Distance* merupakan algoritma yang dibentuk oleh Vladimir Levenshtein pada tahun 1965 sebagai dalam menentukan tingkat kemiripan dua *string* antara satu dengan yang lain. Algoritma ini (atau yang umum dikenal sebagai *Edit Distance*) menentukan tingkat kemiripan sebagai jumlah minimum operasi pengubahan salah satu karakter yang diperlukan untuk mengubah satu *string* menjadi *string* lainnya, dengan operasi pengubahan dinyatakan sebagai salah satu dengan ketiga operasi berikut:

1. Penyisipan (*insertion*): menambah satu karakter;
2. Penghapusan (*deletion*): menghapus satu karakter;
3. Penggantian (*substitution*): mengganti satu karakter dengan yang lain;

Setiap operasi tersebut memiliki biaya sebesar satu untuk dilakukan, dan semakin kecil biaya yang diperlukan untuk melakukan keseluruhan, maka semakin mirip kedua *string* tersebut. Algoritma ini diimplementasikan menggunakan *dynamic programming* yang mempertimbangkan semua kemungkinan proses pengubahan dan menentukan nilai terkecil dalam melakukan pengubahan tersebut. *Dynamic programming* didasarkan pada konsep perbandingan indeks dari kedua *string*. Misalkan *string* pertama dinyatakan dengan variabel kata1 dan indeks *i*, sedangkan *string* kedua dinyatakan dengan variabel kata2 dan indeks *j*, maka

- Jika $\text{kata1}[i]$ dan $\text{kata2}[j]$ sama, maka nilai *Levenshtein* adalah nilai *Levenshtein sub-string* selanjutnya, yaitu $\text{kata1}[i:]$ dan $\text{kata2}[j:]$, atau dengan kata lain memulai proses perbandingan untuk $\text{kata1}[i+1]$ dan $\text{kata2}[j+1]$;
- Jika $\text{kata1}[i]$ dan $\text{kata2}[j]$ berbeda, nilai *Levenshtein* adalah $1 + \text{nilai Levenshtein sub-string}$ selanjutnya, yang dibagi berdasarkan tiga kemungkinan:
 - Operasi penyisipan akan memajukan indeks kata2, sehingga perbandingan selanjutnya menjadi $\text{kata1}[i]$ dan $\text{kata2}[j+1]$;
 - Operasi penghapusan akan memajukan indeks kata1, sehingga perbandingan selanjutnya menjadi $\text{kata1}[i+1]$ dan $\text{kata2}[j]$;
 - Operasi penggantian akan memajukan kedua indeks, sehingga perbandingan selanjutnya menjadi $\text{kata1}[i+1]$ dan $\text{kata2}[j+1]$;

Berdasarkan proses perhitungan tersebut, maka dapat dibentuk matriks sebagai berikut sebagai dasar dalam melakukan *dynamic programming*.

		w 2
	a	c
w 1	a	d
b		
d		

Gambar 6. Matriks representasi nilai *Levensthein Distance* untuk setiap *sub-string*

(Sumber: <https://www.youtube.com/watch?v=XYi2-LPrwm4>)

dengan baris merujuk kepada indeks dari kata1 dan kolom merujuk kepada indeks dari kata2 , untuk kemudian ditambahkan satu baris dan kolom setelahnya. Nilai pada suatu baris dan kolom akan merujuk kepada nilai *Levenshtein Distance* untuk *sub-string* yang ditunjuk oleh $\text{kata1}[\text{baris}:]$ dan $\text{kata2}[\text{kolom}:]$. Sebagai contoh, nilai dari slot matriks pada posisi (2,2), dengan

(1,1) merujuk pada slot paling kiri atas, akan menyatakan nilai *Levenshtein Distance* untuk *sub-string* ‘bd’ dan ‘cd’. Nilai pada baris terakhir dan kolom terakhir diinisiasi dengan *Levenshtein Distance* terhadap salah satu *sub-string* dengan *string* kosong, yaitu melakukan penghapusan sebanyak jumlah huruf pada *sub-string*. Nilai ini digunakan sebagai basis melakukan rekursif secara *bottom-up*.

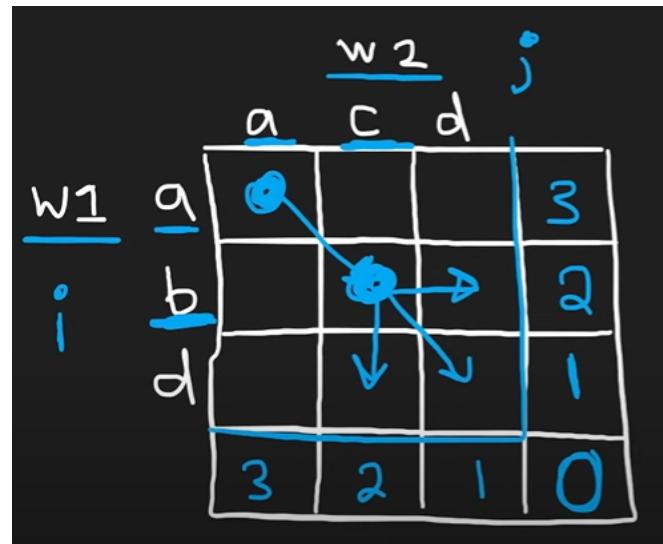
			<u>w2</u>
			a
			c
			d
<u>w1</u>	a		3
b			2
d			1
	3	2	1
			0

Gambar 7. Inisialisasi nilai untuk baris terakhir dan kolom terakhir

(Sumber: <https://www.youtube.com/watch?v=XYi2-LPrwm4>)

Proses *dynamic programming* kemudian melakukan pemberian nilai kepada setiap pasangan *sub-string* berdasarkan metode yang telah disampaikan sebelumnya, dengan realisasi sebagai berikut:

- Jika huruf pada $kata1[\text{baris}]$ dan $kata2[\text{kolom}]$ sama, maka nilai *Levenshtein Distance* adalah nilai dari nilai *Levenshtein Distance* antara $kata1[(\text{baris}+1):]$ dan $kata2[(\text{kolom}+1):]$, yang direpresentasikan oleh nilai pada slot kanan bawah dari slot yang akan diisi;
- Jika huruf pada $kata1[\text{baris}]$ dan $kata2[\text{kolom}]$ berbeda, maka nilai *Levenshtein Distance* adalah nilai minimal antar ketiga kemungkinan operasi yang mungkin, direpresentasikan oleh nilai pada slot kanan, kanan bawah, atau bawah dari slot yang akan diisi, lalu ditambah 1.



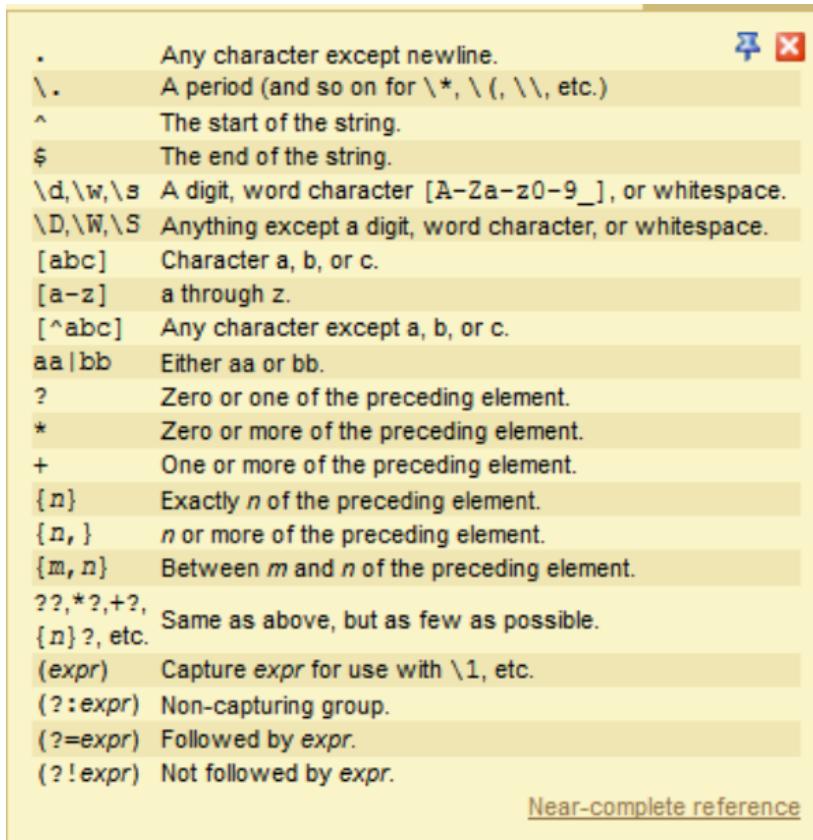
Gambar 8. Tahapan rekursivitas untuk mendapatkan *Levenshtein Distance* antar *string*.

(Sumber: <https://www.youtube.com/watch?v=XYi2-LPrwm4>)

Dengan melakukan rekursif terhadap seluruh *sub-string*, maka akan didapatkan nilai *Levenshtein Distance* antara kedua *string* sebagai nilai yang terletak pada slot paling kiri atas dari matriks.

2.1.5 Regular Expressions (Regex)

Regular Expressions, atau biasa disebut dengan istilah Regex, menyatakan urutan karakter yang membentuk suatu pola dalam melakukan pencarian. Regex menggunakan kumpulan karakter dan metakarakter untuk mendefinisikan pola kompleks yang dicocokan kepada kumpulan *string*. Beberapa contoh metakarakter yang umum digunakan dalam Regex adalah sebagai berikut.



Gambar 9. Kumpulan metakarakter dalam sintaks pencocokan menggunakan Regex

(Sumber: [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))

Python mengintegrasikan penggunaan Regex menggunakan *library* re, dengan contoh implementasi dalam program hasil tugas besar adalah sebagai berikut.

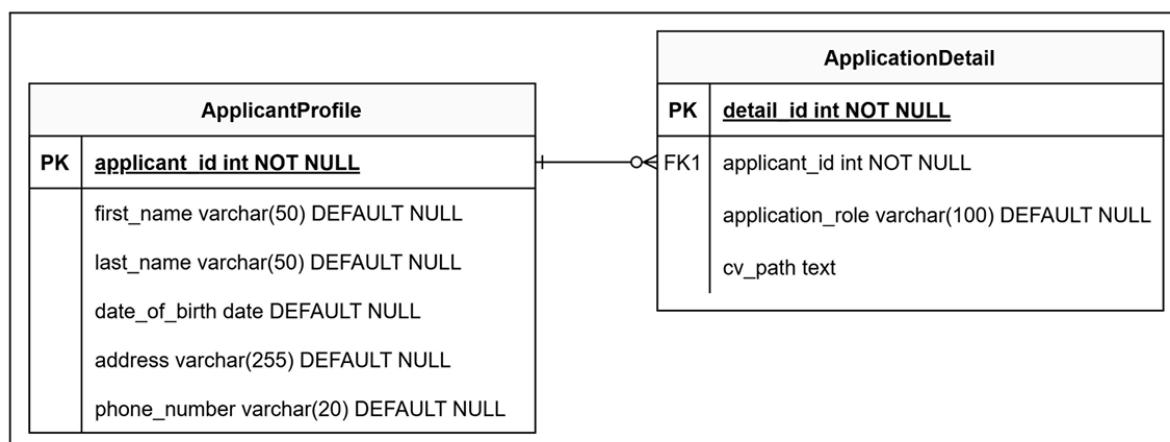
```
skills_lines_regex = re.compile(
    r"[\r\n]+ " + rf"(?:{skills_header_pattern})\s*[:]? \s*[\r\n]+"
    rf"(?P<line1>{VALID_SKILL_LINE_PATTERN})"
    rf"(?:[\r\n]+(?P<line2>{VALID_SKILL_LINE_PATTERN}))?"
    rf"(?:[\r\n]+(?P<line3>{VALID_SKILL_LINE_PATTERN}))?", 
    re.IGNORECASE
)
```

Dalam konteks penyelesaian tugas besar, karena Regex digunakan untuk mengumpulkan *string* sesuai dengan sintaks yang diinginkan, digunakan *capturing group* (bentuk sintaks (?P<...>)) sebagai variabel yang dapat dipanggil dalam mengembalikan *string* yang diinginkan.

2.2 Penjelasan *Database*

Database adalah kumpulan data terorganisir dan terstruktur yang disimpan secara elektronik dalam sistem komputer. *Database* memungkinkan penyimpanan, pengelolaan, dan pengambilan data secara efisien dan sistematis. Secara umum, beberapa kegunaan dari *database* adalah sebagai penyimpanan data terpusat, menghindari redundansi (duplikasi data yang tidak perlu), memastikan integritas data, mengontrol akses dan keamanan data, memudahkan pencadangan dan pemulihan data, memungkinkan *multiple user* mengakses *database* dalam waktu bersamaan, dan sebagainya. Dalam sistem CV ATS, *database* digunakan untuk menyimpan informasi personal mengenai pelamar, detail lamaran, dan file CV yang dilampirkannya secara terorganisir.

Implementasi *database* menggunakan DBMS (*Database Management System*) berupa MySQL. Sistem ini menerapkan 2 tabel utama dalam basis data, yakni tabel ApplicantProfile (detail pelamar) dan ApplicationDetail (detail lamaran). *Database* ini digunakan untuk menyimpan informasi personal seperti nama, tanggal lahir, alamat, dan nomor telepon. *Database* ini juga digunakan untuk menyimpan detail setiap aplikasi pekerjaan termasuk posisi yang dilamar, path lokasi *file* CV untuk setiap aplikasi, menghubungkan data pelamar dengan aplikasi mereka melalui *foreign key*, serta memastikan konsistensi data dengan *constraint* dan validasi. Kemudian, database dihubungkan dengan menggunakan MySQL *connector* yang merupakan *driver* resmi dari Oracle untuk MySQL.



Gambar 10. Skema Basis Data CV ATS

(Sumber: <https://docs.google.com/document/d/1aVJtQ9oj-WE7GrQwaHv3xzYcAMIsPBLAX4SLBcPZOA/edit?tab=t.0>)

2.3 Penjelasan Aplikasi Desktop

Aplikasi desktop yang dibangun adalah aplikasi yang ditujukan untuk meningkatkan *User Experience* (UX) dalam menggunakan program pencarian kata pada sebuah CV. Aplikasi berbasis bahasa pemrograman Python dengan menggunakan *framework* GUI Python, Flet, dan sistem manajemen basis data relasional MySQL. Pengguna dapat mencari kata-kata yang diinginkan pada halaman pencarian. Setiap kata yang berbeda, dipisahkan dengan koma sebagai penanda. Pengguna juga dapat memilih dari tiga pilihan algoritma pencarian yang disediakan, yaitu algoritma Knuth-Morris-Pratt (KMP), algoritma Boyer Moore, dan algoritma Aho Corasick. Setelah itu, pengguna harus memberikan jumlah hasil pencarian yang ingin ditampilkan. Setelah mengisi ketiga hal tersebut, pengguna dapat menekan tombol pencarian. Hasil pencarian akan ditampilkan dengan menggunakan konsep *card*. Data yang ditampilkan berupa nama dan berapa banyak kata pada CV mereka yang cocok dengan *keyword* yang diberikan pengguna. Pada *card* tersebut, terdapat dua tombol, yaitu tombol “summary” dan tombol “view CV” dimana tombol “summary” digunakan untuk melihat ringkasan data dari CV dan tombol “view CV” digunakan untuk membuka CV asli.

BAB 3

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

Berikut adalah langkah-langkah sistem pencocokan dan pencarian informasi pelamar kerja menggunakan algoritma *string matching*, *Levenshtein*, dan Regex:

1. Pre-processing

Pengolahan setiap CV pada direktori data dilakukan melalui fungsi dari *library* yang disediakan oleh Python. Hasil dari proses ekstraksi *file PDF* adalah *dictionary* yang memetakan *string* berupa *path* menuju *file PDF* terhadap *string* berisikan ekstraksi *file PDF* tersebut.

2. *Seeding* dan *encryption database*

Proses *seeding* dan *encryption* di sistem ini dilakukan melalui *method* *setup_encrypted_data()* pada class MySQLDatabaseManager. Sistem melakukan *seeding* data pelamar dengan informasi yang sudah dienkripsi menggunakan metode *caesar cipher* dan *scrambling words* untuk *string*, sementara tanggal lahir dienkripsi menggunakan algoritma khusus dari class Encryption yang membalik tahun, menggeser bulan,, dan menggeser hari. Data yang di-seed mencakup ratusan *record* pelamar yang diberikan dari spesifikasi dan aplikasi kerja mereka dengan *role* dan *path* CV masing-masing.

3. *String-matching*

Menggunakan *dictionary* hasil ekstraksi, setelah pengguna memasukkan masukan berupa *pattern-pattern* yang akan dilakukan *string-matching* dan algoritma *string-matching*, akan dipanggil fungsi yang melakukan salah satu algoritma *string-matching*, baik KMP, BM, maupun Aho-Corasick sesuai masukan pengguna, terhadap seluruh *string* hasil

ekstraksi. Untuk mempercepat proses ini, digunakan *multi-threading* yang membagi proses ke dalam beberapa *thread* dan dilakukan secara paralel.

4. *Fuzzy matches*

Jika sebuah *pattern* tidak memiliki kecocokan sama sekali dengan semua *string* hasil ekstraksi PDF, maka dilakukan *fuzzy matching* berdasarkan *Levenshtein Distance*. Batasan *threshold* dari *fuzzy matching* didasarkan secara dinamis berdasarkan panjang *pattern* untuk memastikan lingkup *fuzzy matches* tidak terlalu sempit maupun terlalu luas.

5. Hasil *String-matching*

Aplikasi kemudian menampilkan informasi yang relevan mengenai hasil *string-matching*, meliputi *file-file* PDF yang memiliki *occurrences* terhadap *pattern*, jumlah kemunculan, tipe kemunculan (*exact / fuzzy*), lama pencarian, dan jumlah PDF yang dilakukan pencocokan. Pengguna kemudian memberikan masukan apakah akan melihat *file* PDF yang telah ditemukan atau *summary* dari *file* PDF tersebut.

6. Integrasi *database* dan algoritma

Basis data dari hasil *seeding* dihubungkan dengan algoritma dan GUI melalui controller. Fungsi di controller ini dipanggil pada bagian kode GUI ketika pengguna menekan tombol ‘search’. Fungsi tersebut pertama-tama akan mencari kata kunci pada seluruh CV yang terhubung dengan seseorang pada basis data. Setelah pencarian selesai, hasil dari fungsi pencarian diolah untuk keperluan GUI. Pengolahan ini menggabungkan data yang berada pada basis data dengan hasil pencarian. Hasil pencarian yang berupa exact match dan fuzzy match juga dipisah sehingga tampilan hasil nantinya menjadi lebih intuitif.

7. Regex

Jika pengguna melihat *summary* dari *file* PDF, maka Regex akan dilakukan terhadap *file* PDF tersebut untuk mencari informasi-informasi yang akan ditunjukkan dalam *summary*, meliputi ringkasan singkat, *skill set*, pendidikan, dan riwayat pekerjaan dari pelamar kerja.

3.2 Proses Pemetaan Masalah

Dalam melakukan *string-matching* dan *Levenshtein Distance*, elemen yang dibutuhkan adalah *pattern* sebagai *string* yang akan dicari jumlah kemunculannya dan teks sebagai *string* yang akan dilakukan pencocokan. Teks juga dibutuhkan oleh Regex sebagai *string* yang akan dicari bagian yang sesuai dengan pola Regex. Sehingga, tahapan dalam memetakan sistem ke dalam permasalahan *string-matching*, *Levenshtein Distance*, dan Regex adalah sebagai berikut:

1. Ekstraksi PDF

Melakukan ekstraksi isi dari setiap *file CV* dalam format PDF sebagai sebuah *string*. Hasil dari ekstraksi berupa *dictionary* yang memetakan *string* dari *path file* tersebut ke isi dari *file*, yang bertujuan untuk menyimpan informasi *path* sebagai kebutuhan *controller* pada aplikasi. Setiap *values* dari *dictionary* ini akan berperan sebagai teks yang dilakukan pencocokan menggunakan algoritma *string-matching* dan Regex.

2. Masukan pengguna

Pattern yang akan dilakukan pencocokan terhadap teks hasil ekstraksi PDF menggunakan *string-matching* akan dimasukan oleh pengguna dalam menjalankan aplikasi. Pengguna akan memasukkan kumpulan *string* pada *page searching* yang dipisahkan oleh koma (sebagai contoh: profesional, dedikasi, SQL), dan diubah menjadi *array of string* ([“profesional”, “dedikasi”, “SQL”]) untuk dapat diproses oleh *string-matching* dan *Levenshtein Distance*.

Untuk Regex, karena proses pencocokan pola bertujuan untuk mengembalikan hal yang sama bagi setiap *file PDF* (meliputi ringkasan, *skill set*, pendidikan, dan riwayat pekerjaan), maka pola Regex di *hard-coded* dan dilakukan pencocokan dengan *values* dari *dictionary* hasil ekstraksi PDF, sehingga tidak dibutuhkan pemetaan dari dinamika sistem.

3.3 Fitur Fungsional dan Arsitektur Aplikasi Desktop

3.3.1 Fitur Fungsional Aplikasi Desktop

Pada aplikasi desktop yang dibangun terdapat beberapa fitur fungsional yang dapat dilakukan. Berikut adalah penjelasan mengenai fitur-fitur tersebut.

- Fitur pencarian

Pada fitur ini, pengguna dapat memasukkan sejumlah kata kunci, jumlah hasil yang ingin ditampilkan, dan memilih algoritma pencarian. Dengan menekan tombol “search”, program akan melakukan pencarian pada sejumlah CV yang ada pada basis data dan folder data. Pencarian dilakukan sesuai dengan algoritma yang dipilih oleh pengguna. Jika pada CV tidak ditemukan pencocokan yang tepat, program akan melakukan *fuzzy matching* dengan menggunakan Levenshtein Distance. Pada kasus dimana pengguna meminta untuk menampilkan lima CV namun hanya ditemukan tiga CV yang cocok (dengan *exact match* dan *fuzzy match*), algoritma program tidak memaksa melakukan *fuzzy match* untuk memenuhi jumlah hasil menjadi lima CV. Program hanya akan menampilkan ketiga CV tersebut. Hal ini berhubungan pula dengan implementasi *fuzzy matching* pada algoritma program.

- Fitur penampilan hasil

Fitur ini terjadi setelah pengguna menggunakan fitur pencarian. Hasil yang didapatkan dari pencarian, akan ditampilkan pada frontend berupa nama pemilik CV dan semua kata kunci yang cocok, baik itu *exact matching* maupun *fuzzy matching*. Hasil pencarian ditampilkan tepat dibawah *field* input yang dapat digulir jika lebih dari tiga hasil ditampilkan (hal ini terjadi karena tiap baris akan menampilkan tiga kolom).

- Fitur menampilkan CV asli

Ketika pengguna mendapatkan hasil dari pencarian, terdapat tombol pada setiap hasil pencarian (per CV) yang dapat digunakan untuk melihat CV asli. CV yang berupa PDF akan ditampilkan dengan mengalihkan ke browser pengguna. Program akan tetap

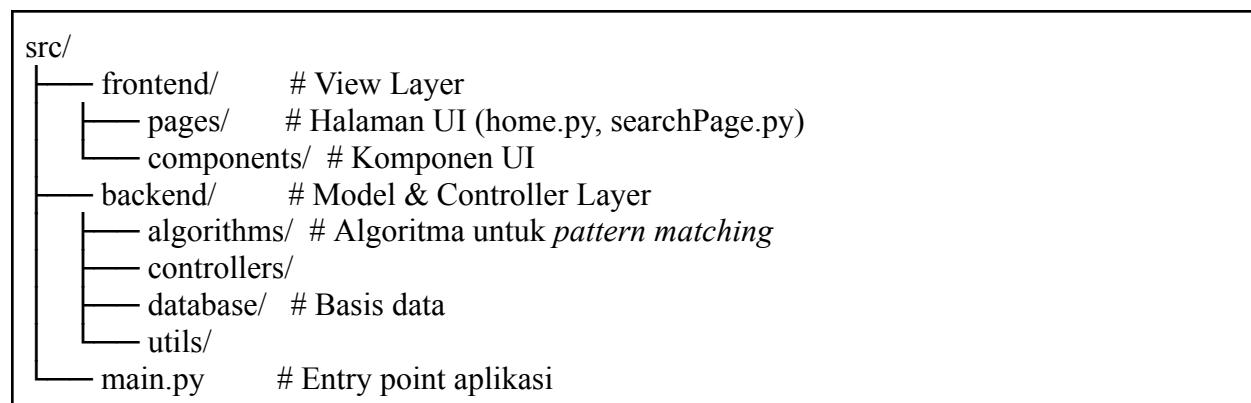
berjalan dalam kondisi ini. Pengguna bisa dengan leluasa menutup browser ataupun melanjutkan kegiatan dengan program entah itu melakukan pencarian maupun membuka CV asli lainnya.

- Fitur ringkasan data CV

Sama seperti fitur sebelumnya, fitur ini dapat digunakan oleh pengguna ketika hasil pencarian sudah ditampilkan. Fitur ini dapat diakses dengan menekan tombol “summary” yang akan membuka komponen dialog. Isi dari dialog adalah data pribadi pengguna yang didapat dari basis data dan data pada CV berupa kemampuan yang dimiliki, pengalaman kerja, dan latar belakang pendidikan.

3.3.2 Arsitektur Aplikasi Desktop

Aplikasi yang dibangun terhubung dengan basis data dimana basis data tersebut menghubungkan setiap CV yang ada pada folder dengan data seseorang. Program dibuat dengan arsitektur Model-View-Controller (MVC). Arsitektur ini memisahkan urusan data, tampilan UI, dan controller untuk penghubung data dan tampilan UI. Pemisahan ini dapat dilihat pada struktur folder yang dimiliki.



Layer *view* berada pada folder *frontend* yang mempunyai folder *pages* dan *components*. Folder *pages* berguna untuk menyimpan file python untuk setiap halaman. Terdapat dua halaman, yaitu halaman utama dan halaman pencarian. Folder *components* berguna untuk menyimpan komponen-komponen UI yang dapat digunakan ulang. Layer *model* dan *controller* terdapat pada folder *backend*. Layer *model* lebih tepatnya terdapat pada folder *database* yang

berguna untuk menyimpan file-file yang mengatur model data, operasi basis data, dan pengaturan koneksi basis data. Layer controller lebih tepatnya terdapat pada folder controllers dan algorithms. Folder algorithms berguna untuk menyimpan implementasi algoritma *pattern matching* sedangkan folder controllers lebih kepada mengatur hubungan antara basis data dan algoritma pencarian untuk kegunaan UI.

Dengan penggunaan MVC, alur data pada aplikasi menjadi seperti berikut ini.

1. Pengguna memasukkan kata kunci, algoritma, dan jumlah *top matches* yang diinginkan.
2. Ketika pengguna menekan tombol “search”, permintaan diteruskan kepada fungsi pencarian pada searchController.py yang ada pada folder controllers.
3. Controller memanggil fungsi stringMatching yang ada pada file algoAPI.py untuk melakukan pencarian.
4. Pencarian dilakukan oleh fungsi stringMatching dan ketika pencarian selesai, hasil dikembalikan kepada controller.
5. Controller mengolah data hasil dengan menggabungkan dengan basis data yang ada. Hal ini berguna untuk menemukan data pribadi untuk setiap CV hasil.
6. Data gabungan tersebut dikembalikan pada layer view dan layer view menampilkan hasil ke pengguna.

3.4 Contoh Ilustrasi Kasus

Contoh alur penggunaan program adalah sebagai berikut:

1. Pengguna menjalankan program, lalu masuk ke *tab search* untuk melakukan pencocokan;
2. Pengguna memasukkan kata-kata yang ingin dicari pada keseluruhan *file* CV, dengan setiap kata dipisahkan oleh koma;
3. Pengguna memasukkan masukan-masukan yang diperlukan dalam melakukan pencocokan, meliputi algoritma pencarian dan jumlah CV yang ingin ditampilkan;
4. Pengguna melakukan pencarian, dan program akan menampilkan CV-CV yang memenuhi masukan pengguna beserta informasi mengenai pencarian;
 - a. Informasi mengenai hasil pencarian akan bervariasi berdasarkan masukan pengguna, jika terdapat *typo* yang mengakibatkan tidak terdapat *exact matches*

terhadap masukan pengguna, maka akan dilakukan *fuzzy matches*. Informasi hasil pencarian akan mengutamakan *exact matches* dibandingkan *fuzzy matches*.

5. Pengguna dapat melihat *summary* dari *file* CV hasil pencocokan, yang akan menampilkan ringkasan singkat, *skill set*, pendidikan, dan riwayat pendidikan dari *file* CV tersebut berdasarkan hasil pencocokan Regex;
6. Pengguna juga dapat melihat *file* CV hasil pencocokan, yang akan menampilkan *file* PDF dari CV tersebut.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Struktur Data

```
CREATE TABLE ApplicantProfile (
    applicant_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE,
    address VARCHAR(255),
    phone_number VARCHAR(20)
)
```

Tabel ini berisi struktur data pelamar yang terdiri dari empat kolom utama, yakni id sebagai *primary key* bertipe *integer* dengan *auto-increment* untuk identifikasi unik setiap pelamar, first_name dan last_name bertipe VARCHAR yang menyimpan nama depan dan belakang pelamar dalam bentuk terenkripsi, date_of_birth bertipe DATE yang menyimpan tanggal lahir pelamar yang telah dienkripsi, serta address (VARCHAR) sebagai alamat dan phone_number (VARCHAR) yang juga dienkripsi untuk menjaga keamanan data pribadi pelamar.

```
CREATE TABLE ApplicationDetail (
    detail_id INT AUTO_INCREMENT PRIMARY KEY,
    applicant_id INT NOT NULL,
    application_role VARCHAR(100),
    cv_path TEXT,
    FOREIGN KEY (applicant_id) REFERENCES ApplicantProfile(applicant_id)
)
```

Tabel ini berisi struktur data detail lamaran pekerjaan yang diajukan oleh setiap pelamar dengan struktur relasional yang terdiri dari empat kolom, yakni id sebagai *primary key* bertipe *integer* dengan *auto-increment* untuk identifikasi unik setiap lamaran, applicant_id bertipe *integer* sebagai *foreign key* yang mereferensikan kolom id pada tabel ApplicantProfile untuk membuat hubungan *one-to-many* antara pelamar dan lamaran pekerjaan mereka, *role* bertipe

VARCHAR yang menyimpan nama posisi atau bidang pekerjaan yang dilamar, dan cv_path bertipe VARCHAR yang menyimpan *path file* CV pelamar.

4.2 Fungsi dan Prosedur Program

Algoritma:

aho_corasick.py

Fungsi/Prosedur	Penjelasan
<pre>def __init__(self): self.children = {} self.output = [] self.failure_link = None self.parent = None self.char_from_parent = None</pre>	Fungsi ini bertujuan untuk menginisialisasi <i>node</i> dalam struktur Trie.
<pre>def __init__(self, patterns): self.root = TrieNode() self.patterns = patterns self._build_trie() self._build_failure_links()</pre>	Fungsi ini bertujuan untuk inisialisasi <i>automaton</i> algoritma Aho-Corasick.
<pre>def _build_trie(self): """ Membangun struktur Trie dari semua pola yang diberikan (Fase Go-to). """ for i, pattern in enumerate(self.patterns): current_node = self.root for char in pattern: if char not in current_node.children: new_node = TrieNode() new_node.parent = current_node new_node.char_from_parent = char</pre>	Fungsi ini bertujuan untuk membangun struktur Trie dengan hasil <i>node</i> akhir mewakili <i>prefix</i> .

<pre> current_node.children[char] = new_node current_node = current_node.children[char] current_node.output.append(i) </pre>	
<pre> def _build_failure_links(self): """ Membangun failure links untuk setiap node dalam Trie menggunakan BFS. """ queue = collections.deque() for char, child_node in self.root.children.items(): child_node.failure_link = self.root queue.append(child_node) while queue: current_node = queue.popleft() for char, next_node in current_node.children.items(): failure_state = current_node.failure_link while failure_state is not None and char not in failure_state.children: if failure_state == self.root: failure_state = None break failure_state = failure_state.failure_link if failure_state is None: next_node.failure_link = self.root </pre>	<p>Fungsi ini bertujuan untuk membangun <i>failure links</i> untuk memungkinkan transisi ketika terdapat karakter yang tidak cocok.</p>

```

        else:

next_node.failure_link =
failure_state.children[char]

next_node.output.extend(next_node.failure_link.output)

queue.append(next_node)

```

```

def search(self, text):
    """
        Mencari semua kemunculan pola
        dalam teks menggunakan automaton
        Aho-Corasick
        dan mengembalikan array of
        integer yang berkorelasi dengan jumlah
        kemunculan
        setiap pola sesuai dengan
        urutan pola yang diberikan.
    """

```

Args:

text (str): Teks input
yang akan dipindai.

Returns:

list: Array of integer, di
mana setiap elemen adalah jumlah
kemunculan
dari pola yang
sesuai (berdasarkan indeks
`self.patterns`).

```

    pattern_counts = [0] *
len(self.patterns)

    current_state = self.root

    for i, char in
enumerate(text):
        while current_state is not
None and char not in
current_state.children:
            if current_state ==

```

Fungsi ini bertujuan untuk mencari semua kemunculan *patterns* dalam *text*, mengembalikan *array integer* berisi jumlah kemunculan setiap *pattern*.

```

self.root:
    current_state =
None
        break
    current_state =
current_state.failure_link

    if current_state is None:
        current_state =
self.root
    else:
        current_state =
current_state.children[char]

    for pattern_idx in
current_state.output:

pattern_counts[pattern_idx] += 1

return pattern_counts

```

```

def search_aho(patterns, text):
    ac_automaton =
AhoCorasick(patterns)
    return ac_automaton.search(text)

```

Fungsi ini bertujuan sebagai API sederhana untuk menggunakan algoritma Aho-Corasick.

algoAPI.py

Fungsi/Prosedur	Penjelasan
<pre> def stringMatching(patterns: list[str], algorithm: int): if not (0 <= algorithm <= 2): raise ValueError("Invalid algorithm value. Use 0 for KMP, 1 for Boyer-Moore, or 2 for Aho-Corasick.") # dir_path = os.path.join(os.path.dirname(__file__), , '..', '..', 'data', '11152490.pdf') </pre>	<p>Fungsi ini bertujuan untuk melakukan validasi parameter algoritma, kemudian mengekstrak teks dari semua file PDF di folder data menggunakan <i>multiprocessing</i>, lalu memilih algoritma pencarian sesuai parameter. Setelah itu semua CV diproses secara paralel untuk mencari <i>exact match</i> dari <i>patterns</i> yang diberikan, hasilnya disimpan dalam struktur data dengan <i>counter</i> dan <i>match flags</i>. Kemudian, dilakukan <i>Fuzzy Matching</i> yang hanya memproses <i>patterns</i> yang tidak ditemukan <i>exact match</i>-nya, menggunakan</p>

```

        current_dir =
os.path.dirname(os.path.abspath(__file__))
        project_root =
os.path.abspath(os.path.join(current_d
ir, '.', '.', '.'))
        data_dir =
os.path.join(project_root, 'data')

        if not os.path.exists(data_dir):
            print(f"Folder data tidak
ditemukan: {data_dir}")
            return None
        print(f"Mengekstrak PDF dari
direktori: {data_dir}")
        dict_of_cv_texts =
extract_pdfs(data_dir,
max_workers=os.cpu_count())

        if not dict_of_cv_texts:
            print("Tidak ada teks CV yang
berhasil diekstrak. Menghentikan
proses.")
            return None

        print(f"Berhasil mengekstrak
{len(dict_of_cv_texts)} CV.")

        search_algorithm_func = None
        if algorithm == 0:
            search_algorithm_func =
search_kmp
        elif algorithm == 1:
            search_algorithm_func =
search_bm
        else:
            search_algorithm_func =
search_aho

        print("Memulai pencarian exact
matching...")
        results_per_cv_akumulatif = {}
        exact_cv_processed_count = 0

```

algoritma Levenshtein untuk mencari *similarity match* dengan pendekatan *task-per-combination* (setiap CV-pattern menjadi task terpisah) yang juga dijalankan secara paralel. Terakhir, hasil dari kedua phase digabungkan menjadi *output* final berupa *dictionary* yang berisi *counter* (jumlah kemunculan) dan *match flags* (status: 0 = tidak ada, 1 = exact, 2 = fuzzy) untuk setiap CV beserta *metrics* waktu eksekusi dan jumlah CV yang diproses.

```

        start_exact_time =
time.perf_counter()

        with
concurrent.futures.ThreadPoolExecutor(
max_workers=os.cpu_count()) as
executor:
    def
_run_exact_search_on_cv(cv_item_tuple)
:
    cv_name, cv_text =
cv_item_tuple
        exact_counts_for_patterns
= search_algorithm_func(patterns,
cv_text)
        return cv_name,
exact_counts_for_patterns

    exact_futures =
[executor.submit(_run_exact_search_on_
cv, item)
        for item in
dict_of_cv_texts.items()]

    for future in
concurrent.futures.as_completed(exact_
futures):
        exact_cv_processed_count
+= 1
        cv_path,
exact_counts_for_current_cv =
future.result()

results_per_cv_akumulatif[cv_path] = {
        "counter": [0] *
len(patterns),
        "match_flags": [0] *
len(patterns),
    }

    for i, count in
enumerate(exact_counts_for_current_cv)

```

```

:

results_per_cv_akumulatif[cv_path][ "co
unter"] [i] = count
    if count > 0:

results_per_cv_akumulatif[cv_path][ "ma
tch_flags"] [i] = 1

    end_exact_time =
time.perf_counter()
    exact_time = end_exact_time -
start_exact_time

    print(f"Exact matching selesai.
{exact_cv_processed_count} CVs
diproses dalam {exact_time:.4f}
detik.")

    print("Memulai pencarian fuzzy
matching...")
    fuzzy_cv_processed_count = 0
    all_fuzzy_tasks = []

    start_fuzzy_time =
time.perf_counter()

    for cv_path, cv_text in
dict_of_cv_texts.items():

needs_fuzzy_processing_for_this_cv =
False

    patterns_for_fuzzy_this_cv =
[]
        for i in range(len(patterns)):
            if
results_per_cv_akumulatif[cv_path][ "co
unter"] [i] == 0:

patterns_for_fuzzy_this_cv.append(patt
erns[i])

```

```

needs_fuzzy_processing_for_this_cv =
True

    if
needs_fuzzy_processing_for_this_cv:
        fuzzy_cv_processed_count
+= 1

            for pattern_item in
patterns_for_fuzzy_this_cv:

all_fuzzy_tasks.append((cv_path,
pattern_item, cv_text))

    if all_fuzzy_tasks:
        with
concurrent.futures.ThreadPoolExecutor(
max_workers=os.cpu_count()) as
fuzzy_executor:
            def
_run_fuzzy_search_on_single_pattern_in
_cv(task_tuple):
                cv_path, pattern_item,
cv_text = task_tuple

fuzzy_count_for_pattern =
count_fuzzy(pattern_item, cv_text)
                return cv_path,
pattern_item, fuzzy_count_for_pattern

            fuzzy_futures =
[fuzzy_executor.submit(_run_fuzzy_sear
ch_on_single_pattern_in_cv, task) for
task in all_fuzzy_tasks]

            for future in
concurrent.futures.as_completed(fuzzy_
futures):
                cv_path,
fuzzy_pattern, fuzzy_count_for_pattern
= future.result()

            if

```

```

fuzzy_count_for_pattern > 0:
    idx =
patterns.index(fuzzy_pattern)

results_per_cv_akumulatif[cv_path][ "co
unter"] [idx] = fuzzy_count_for_pattern

results_per_cv_akumulatif[cv_path][ "ma
tch_flags"] [i] = 2

    end_fuzzy_time =
time.perf_counter()
    fuzzy_time = end_fuzzy_time -
start_fuzzy_time
    print(f"Fuzzy matching selesai.
{fuzzy_cv_processed_count} CVs
diproses dalam {fuzzy_time:.4f}
detik.")

    print("Menyusun hasil akhir...")
    final_output_results = {}
    for cv_path, data in
results_per_cv_akumulatif.items():
        final_output_results[cv_path]
= (
            data[ "counter"],
            data[ "match_flags"]
        )

    return (final_output_results,
exact_time, fuzzy_time,
exact_cv_processed_count,
fuzzy_cv_processed_count)

```

boyer_moore.py

Fungsi/Prosedur	Penjelasan
<pre> def build_last(pattern): """ Membangun tabel kemunculan </pre>	Fungsi ini bertujuan untuk membangun tabel <i>last occurrence</i> untuk algoritma Boyer-Moore.

```

terakhir (last occurrence) untuk
algoritma Boyer-Moore.

"""

last = [-1] * 128
for i in range(len(pattern)):
    last[ord(pattern[i])] = i
return last

```

```

def bm_search_single(text, pattern):
    """
    Melakukan pencarian satu pola
    string menggunakan algoritma
    Boyer-Moore (non-overlapping).

    """

    n = len(text)
    m = len(pattern)

    if m == 0:
        return 0
    if n == 0:
        return 0
    if m > n:
        return 0

    last = build_last(pattern)
    count = 0
    i = m - 1

    if i > n - 1:
        return 0

    while i < n:
        j = m - 1
        k = i

        while j >= 0 and text[k] == pattern[j]:
            k -= 1
            j -= 1

        if j < 0:
            count += 1
            i += m
        else:

```

Fungsi ini bertujuan untuk melakukan pencarian Boyer-Moore untuk satu *pattern* dan akan mengembalikan *integer* berisi jumlah kemunculan *pattern* (*non-overlapping*).

```

        text_char = text[k]
        lo = last[ord(text_char)]
if ord(text_char) < 128 else -1
        shift = max(1, j - lo)
        i += shift

    return count

```

```

def bm_search(text, patterns,
max_workers=None):
    """
        Melakukan pencarian beberapa pola
        string menggunakan algoritma
        Boyer-Moore (non-overlapping)
        dengan multithreading.
    """

    Args:
        text (str): String teks yang
        akan dicari polanya.
        patterns (list[str]): List
        string pola yang akan dicari.
        max_workers (int, optional):
        Jumlah thread maksimum yang akan
        digunakan.
    
```

Jika None, default-nya adalah jumlah CPU.

```

    Returns:
        list[int]: Array of integer,
        di mana setiap elemen adalah jumlah
        kemunculan
                dari pola yang
        sesuai (berdasarkan indeks `patterns`
        input).
    """

    text_lower = text.lower()
    pattern_counts = [0] *
len(patterns)
    with
concurrent.futures.ThreadPoolExecutor(
max_workers=max_workers) as executor:
        futures =
{executor.submit(bm_search_single,

```

Fungsi ini bertujuan untuk API *wrapper* untuk Boyer-Moore *search* yang memanggil *bm_search()* dengan parameter *default*.

```

text_lower, pattern.lower()): idx
        for idx, pattern in
enumerate(patterns)}

        for future in
concurrent.futures.as_completed(future
s):
            idx = futures[future]
            try:
                result =
future.result()
                pattern_counts[idx] =
result
            except Exception as exc:
                print(f"Pola
{patterns[idx]} menghasilkan
kesalahan: {exc}")

    return pattern_counts

```

```

def search_bm(patterns, text):
    return bm_search(text, patterns)

```

Fungsi ini bertujuan sebagai API sederhana untuk menggunakan algoritma Boyer-Moore.

kmp.py

Fungsi/Prosedur	Penjelasan
<pre> def compute_lps_array(pattern): """ Menghitung array LPS (longest proper prefix suffix) untuk algoritma KMP. Array LPS menyimpan panjang awalan terpanjang yang juga merupakan akhiran. ex: abcabc --> [0, 0, 0, 1, 2, 3] """ m = len(pattern) lps = [0] * m length = 0 i = 1 </pre>	<p>Fungsi ini bertujuan untuk menghitung <i>array LPS</i> untuk algoritma KMP.</p>

```

        while i < m:
            if pattern[i] ==
pattern[length]:
                length += 1
                lps[i] = length
                i += 1
            else:
                if length != 0:
                    length = lps[length -
1]
                else:
                    lps[i] = 0
                    i += 1
    return lps

```

```

def kmp_search_single(text, pattern):
    """
    Melakukan pencarian satu pola
    string menggunakan algoritma
    Knuth-Morris-Pratt (KMP).

```

Args:

- text (str): String teks yang akan dicari polanya.
- pattern (str): String pola yang akan dicari.

Returns:

- int: Jumlah kemunculan pola dalam teks (non-overlapping).

```

n = len(text)
m = len(pattern)
```

```

if m == 0:
    return 0
if n == 0:
    return 0
```

```

lps = compute_lps_array(pattern)
count = 0
i = 0
j = 0
```

Fungsi ini bertujuan untuk mencari sebuah *pattern* menggunakan algoritma KMP dan mengembalikan *integer* jumlah kemunculan *pattern* (*non-overlapping*).

```

while i < n:
    if pattern[j] == text[i]:
        i += 1
        j += 1

    if j == m:
        count += 1
        j = lps[j - 1]
    elif i < n and pattern[j] != text[i]:
        if j != 0:
            j = lps[j - 1]
        else:
            i += 1
return count

```

```

def kmp_search_multi_threaded(text,
patterns, max_workers=None):
    """
    Melakukan pencarian beberapa pola
    string menggunakan algoritma
    Knuth-Morris-Pratt (KMP)
    dengan multithreading.

    Args:
        text (str): String teks yang
        akan dicari polanya.
        patterns (list[str]): List
        string pola yang akan dicari.
        max_workers (int, optional):
        Jumlah thread maksimum yang akan
        digunakan.

    Jika None, default-nya adalah jumlah
    CPU.

    Returns:
        list[int]: Array of integer,
        di mana setiap elemen adalah jumlah
        kemunculan
                dari pola yang
        sesuai (berdasarkan indeks `patterns`
        input).
    """

```

Fungsi ini bertujuan untuk mencari *multiple patterns* menggunakan KMP dengan *multithreading*.

```

text_lower = text.lower()
pattern_counts = [0] *
len(patterns)
with concurrent.futures.ThreadPoolExecutor(
max_workers=max_workers) as executor:
    futures =
{executor.submit(kmp_search_single,
text_lower, pattern.lower()): idx
for idx, pattern in enumerate(patterns)}
    for future in concurrent.futures.as_completed(futures):
        idx = futures[future]
        try:
            result =
future.result()
            pattern_counts[idx] =
result
        except Exception as exc:
            print(f"Pola
{patterns[idx]} menghasilkan
kesalahan: {exc}")
return pattern_counts

```

```

def search_kmp(patterns, text):
    return kmp_search_multi_threaded(text,
patterns)

```

Fungsi ini bertujuan sebagai API sederhana untuk menggunakan algoritma Knuth-Morris-Pratt.

levenshtein.py

Fungsi/Prosedur	Penjelasan
<pre> def minDistance(self, word1: str, word2: str) -> int: cache = [[float("inf")] * (len(word2) + 1) for _ in range(len(word1) + 1)] for j in range(len(word2)) + </pre>	Fungsi ini bertujuan untuk menghitung jarak Levenshtein antara dua <i>string</i> .

```

1):
    cache[len(word1)][j] =
len(word2) - j
    for i in range(len(word1) +
1):
        cache[i][len(word2)] =
len(word1) - i
        for i in range(len(word1) - 1,
-1, -1):
            for j in range(len(word2)
- 1, -1, -1):
                if word1[i] ==
word2[j]:
                    cache[i][j] =
cache[i + 1][j + 1]
                else:
                    cache[i][j] = 1 +
min(cache[i + 1][j], cache[i][j + 1],
cache[i + 1][j + 1])

return cache[0][0]

```

```

def countThreshold(self, word: str) ->
int:
    """
        Menghitung threshold untuk
jarak Levenshtein.
        Threshold adalah panjang
string dibagi 2, dibulatkan ke atas.
    """
    length = len(word)
    if length <= 3:
        return 0
    elif 3 < length <= 6:
        return 1
    elif 6 < length <= 10:
        return 2
    else:
        return 3

```

```

def countFuzzy(self, pattern, text) ->
int:
    threshold =
self.countThreshold(pattern)

```

Fungsi ini bertujuan untuk menentukan *threshold* toleransi untuk *fuzzy matching* berdasarkan panjang kata.

Fungsi ini bertujuan untuk menghitung jumlah *fuzzy matches* dari *pattern* dalam sebuah *text*.

```

fuzzy_count = 0
pattern_lower =
pattern.lower()
text_lower = text.lower()
words_in_text =
re.findall(r'\b\w+\b', text_lower)

    for word_in_cv in
words_in_text:
        distance =
self.minDistance(pattern_lower,
word_in_cv)
        if distance <= threshold:
            fuzzy_count += 1

    return fuzzy_count

```

```

def count_fuzzy(pattern, text):
    """
        Fungsi untuk menghitung jumlah
        fuzzy match dari pattern dalam teks.
        Ini adalah fungsi pembungkus untuk
        memudahkan penggunaan.
    """
    sol = Solution()
    return sol.countFuzzy(pattern,
text)

```

Fungsi ini bertujuan sebagai *wrapper function* untuk memudahkan penggunaan dengan membuat *instance* Solution dan memanggil fungsi countFuzzy().

Database:

connection.py

Fungsi/Prosedur	Penjelasan
<pre> def __init__(self, host="localhost", user="root", password="", database="cv_analyzer"): self.host = host self.user = user self.password = password self.database = database </pre>	Fungsi ini bertujuan untuk menginisialisasi parameter koneksi pada <i>database</i> .

<pre>self.connection = None</pre>	
<pre>def create_database_if_not_exists(self): try: connection = mysql.connector.connect(host=self.host, user=self.user, password=self.password) cursor = connection.cursor() cursor.execute(f"CREATE DATABASE IF NOT EXISTS {self.database}") print(f"Database '{self.database}' created or already exists") cursor.close() connection.close() except Error as e: print(f"Error creating database: {e}")</pre>	Fungsi ini bertujuan untuk membuat <i>database</i> jika belum ada dan diberi nama cv_analyzer.
<pre>def create_tables(self): connection = self.get_connection() if not connection: return False try: cursor = connection.cursor() create_applicants_table = """ CREATE TABLE IF NOT EXISTS ApplicantProfile (applicant_id INT AUTO_INCREMENT PRIMARY KEY, first_name VARCHAR(50), last_name VARCHAR(50), date_of_birth DATE,</pre>	Fungsi ini bertujuan untuk membuat tabel yang diperlukan untuk aplikasi ini, yakni tabel ApplicantProfile dan ApplicationDetail.

```

        address VARCHAR(255),
        phone_number
VARCHAR(20)
    )
"""

create_applications_table
= """
        CREATE TABLE IF NOT EXISTS
ApplicationDetail (
            detail_id INT
AUTO_INCREMENT PRIMARY KEY,
            applicant_id INT NOT
NULL,
            application_role
VARCHAR(100),
            cv_path TEXT,
            FOREIGN KEY
(applicant_id) REFERENCES
ApplicantProfile(applicant_id)
        )
"""

cursor.execute(create_applicants_table
)

cursor.execute(create_applications_table
)
connection.commit()
print("Database tables
created successfully")
return True
except Error as e:
    print(f"Error creating
tables: {e}")
    return False
finally:
    cursor.close()

```

```

def initialize_connection(self):
    try:

self.create_database_if_not_exists()
    if self.create_tables():
        print("Database

```

Fungsi ini bertujuan untuk menginisialisasi seluruh *setup database* dengan memanggil fungsi `create_database_if_not_exist()` dan `create_tables()` secara berurutan.

```

connection initialized successfully")
        return True
    else:
        return False

except Exception as e:
    print(f"Error initializing
database connection: {e}")
    return False

```

```

def get_connection(self):
    try:
        if self.connection is None
or not self.connection.is_connected():
            self.connection =
mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
        return self.connection
    except Error as e:
        print(f"Error connecting
to MySQL: {e}")
        return None

```

```

def close_connection(self):
    if self.connection and
self.connection.is_connected():
        self.connection.close()
        print("MySQL connection
closed")

```

```

def is_database_setup(self):
    """ Check apakah database
sudah disiapkan dengan data """
    try:
        connection =
self.get_connection()
        if not connection:
            return False

```

Fungsi ini bertujuan untuk mendapatkan koneksi ke *database* yang aktif.

Fungsi ini bertujuan untuk menutup koneksi ke *database* yang aktif.

Fungsi ini bertujuan untuk mengecek apakah *database* sudah terisi data atau belum.

```

        cursor =
connection.cursor()
        cursor.execute("SELECT
COUNT(*) FROM ApplicantProfile")
        result = cursor.fetchone()
        cursor.close()

        if result and result[0] >
0: # type: ignore
            return True
        return False
    except Error:
        return False
    except Exception:
        return False

```

```

def setup_encrypted_data(self):
    """ Setup database dengan
encrypted data dari semua pelamar di
database.sql """
    try:
        from .operations import
DatabaseOperations
        from datetime import
datetime

        print("Setting up database
with encrypted data from
database.sql...")
        all_applicants = [
            (1, 'Moh4mm4d',
'Nu9r4h4', '2003-06-14', 'Jl. Kenanga
No. 12, Jakarta', '081234567891'),
            (2, 'MOH4MM4D',
'NUGR4H4', '2004-03-22', 'Jl. Melati
No. 45, Bandung', '082123456781'),
            (3, 'M0hammad',
'Nugr4h4', '2003-11-05', 'Jl. Cemara
No. 7, Surabaya', '081345678912'),
            (4, 'Mohammad',
'NUGR4H4', '2004-01-17', 'Jl. Sakura
No. 4, Semarang', '082134567892'),
            (5, 'm0h4mm4d',

```

Fungsi ini bertujuan untuk melakukan *setup* pada *database* dengan data pelamar yang telah dienkripsi.

```

'nu9r4h4', '2003-09-12', 'Jl. Mawar
No. 9, Yogyakarta', '081223456789'),
]

applicant_id_mapping = {}
created_count = 0

print(f"Creating
{len(all_applicants)} encrypted
applicants...")

for old_id, first_name,
last_name, dob, address, phone in
all_applicants:
    try:
        dob_date =
datetime.strptime(dob,
'%Y-%m-%d').date()
        applicant_id =
DatabaseOperations.create_applicant(
            first_name =
first_name,
            last_name =
last_name,
            date_of_birth
= dob_date,
            address =
address,
            phone_number =
phone
        )
        if applicant_id:

applicant_id_mapping[old_id] =
applicant_id
            created_count
+= 1
            if created_count %
10 == 0:

print(f"Created {created_count}
encrypted applicants...")
        except Exception as e:

```

```

                print(f"Error
creating applicant {first_name}
{last_name}: {e}")

        print(f"Successfully
created {created_count} encrypted
applicants.")

        all_applications = [
            (1, 20, 'Software
Developer',
'data/INFORMATION-TECHNOLOGY/15118506.
pdf'),
            (2, 27, 'Financial
Planner',
'data/FINANCE/12858898.pdf'),
            (3, 74, None,
'data/CHEF/11121498.pdf'),
            (4, 13, 'Aviation
Mechanic',
'data/AVIATION/11169163.pdf'),
            (5, 42, 'Sales
Consultant',
'data/SALES/15273850.pdf'),
        ]

        app_count = 0
        print(f"Creating
{len(all_applications)}
applications...")

        for detail_id,
old_applicant_id, role, cv_path in
all_applications:
            if old_applicant_id in
applicant_id_mapping:
                try:
                    app_id =
DatabaseOperations.create_application(
applicant_id=applicant_id_mapping[old_
applicant_id],

```

```

application_role=role,
cv_path=cv_path
)
if app_id:
    app_count
+= 1
    if
app_count % 10 == 0:
    print(f"Created {app_count}
applications...")
except Exception
as e:
    print(f"Error
creating application {role}: {e}")
else:
    print(f"Skipping
application for applicant_id
{old_applicant_id} because it was not
found in mapping")

    print(f"Successfully
created {app_count} applications!")
    print("Database setup with
encrypted data completed.")
    return True

except Exception as e:
    print(f"Error setting up
encrypted data: {e}")
    return False

```

```

def get_db_connection():
    return db_manager.get_connection()

```

Fungsi ini merupakan fungsi *helper* yang menjadi *wrapper* untuk db_manager.get_connection().

```

def create_connection(host_name,
user_name, user_password, db_name):
    connection = None
    try:
        connection =
mysql.connector.connect(
            host=host_name,

```

Fungsi ini bertujuan untuk membuat koneksi ke *database MySQL*.

```

        user=user_name,
        password=user_password,
        database=db_name
    )
    print("Connection to MySQL DB
successful")
except Error as e:
    print(f"The error '{e}' occurred")

return connection

```

models.py

Fungsi/Prosedur	Penjelasan
<pre> def __init__(self, applicant_id: Optional[int] = None, first_name: Optional[str] = None, last_name: Optional[str] = None, date_of_birth: Optional[date] = None, address: Optional[str] = None, phone_number: Optional[str] = None): self.applicant_id = applicant_id self.first_name = first_name self.last_name = last_name self.date_of_birth = date_of_birth self.address = address self.phone_number = phone_number </pre>	Fungsi ini bertujuan untuk menginisialisasi objek profil pelamar dan menyimpan informasi personal pelamar tersebut dalam bentuk objek Python.
<pre> # properties dengan encryption/decryption @property def first_name(self) -> </pre>	Fungsi-fungsi ini merupakan properti untuk melakukan enkripsi pada <i>database</i> .

```

Optional[str]:
    return
decrypt(self._first_name) if
self._first_name else None

    @first_name.setter
    def first_name(self, value:
Optional[str]):
        self._first_name =
encrypt(value) if value else None

    @property
    def last_name(self) ->
Optional[str]:
    return
decrypt(self._last_name) if
self._last_name else None

    @last_name.setter
    def last_name(self, value:
Optional[str]):
        self._last_name =
encrypt(value) if value else None

    @property
    def date_of_birth(self) ->
Optional[date]:
    if self._date_of_birth:
        if
isinstance(self._date_of_birth, (date,
datetime)):
            print(f"Date already a
date object: {self._date_of_birth}")
            if
isinstance(self._date_of_birth,
datetime):
                return
decrypt_date(self._date_of_birth.date(
))
                return
decrypt_date(self._date_of_birth.isofo
rmat())
            else:

```

```

        print(f"Unexpected
date_of_birth type:
{type(self._date_of_birth)}")
            return None
        return None

    @date_of_birth.setter
    def date_of_birth(self, value:
Optional[date]):
        if value:
            date_str =
value.isoformat()
            self._date_of_birth =
encrypt_date(date_str)
        else:
            self._date_of_birth = None

    @property
    def address(self) ->
Optional[str]:
        return decrypt(self._address)
if self._address else None

    @address.setter
    def address(self, value:
Optional[str]):
        self._address = encrypt(value)
if value else None

    @property
    def phone_number(self) ->
Optional[str]:
        return
decrypt(self._phone_number) if
self._phone_number else None

    @phone_number.setter
    def phone_number(self, value:
Optional[str]):
        self._phone_number =
encrypt(value) if value else None

    # metode untuk database operations

```

```
(encrypted_data)
    def get_encrypted_first_name(self)
-> Optional[str]:
    return self._first_name

    def get_encrypted_last_name(self)
-> Optional[str]:
    return self._last_name

    def
get_encrypted_date_of_birth(self) ->
Optional[str]:
    return self._date_of_birth

    def get_encrypted_address(self) ->
Optional[str]:
    return self._address

    def
get_encrypted_phone_number(self) ->
Optional[str]:
    return self._phone_number

    def set_encrypted_first_name(self,
encrypted_value: Optional[str]):
        self._first_name =
encrypted_value

    def set_encrypted_last_name(self,
encrypted_value: Optional[str]):
        self._last_name =
encrypted_value

    def
set_encrypted_date_of_birth(self,
encrypted_value: Optional[str]):
        self._date_of_birth =
encrypted_value

    def set_encrypted_address(self,
encrypted_value: Optional[str]):
        self._address =
encrypted_value
```

<pre> def set_encrypted_phone_number(self, encrypted_value: Optional[str]): self._phone_number = encrypted_value </pre>	
<pre> def to_dict(self): """ Convert ke dictionary untuk JSON """ return { 'applicant_id': self.applicant_id, 'first_name': self.first_name, 'last_name': self.last_name, 'date_of_birth': self.date_of_birth.isoformat() if self.date_of_birth is not None else None, 'address': self.address, 'phone_number': self.phone_number } </pre>	<p>Fungsi ini bertujuan untuk mengkonversi objek menjadi <i>dictionary</i> untuk memudahkan dalam API <i>response</i>.</p>
<pre> @classmethod def from_dict(cls, data: Dict[str, Any]): """ Buat instance dari dictionary """ obj = cls() obj.applicant_id = data.get('applicant_id') obj.first_name = data.get('first_name') obj.last_name = data.get('last_name') obj.address = data.get('address') obj.phone_number = data.get('phone_number') if data.get('date_of_birth'): </pre>	<p>Fungsi ini bertujuan untuk membuat <i>instance</i> ApplicantProfile dari <i>dictionary</i> (deserialisasi data dari JSON/<i>dictionary</i>).</p>

<pre> if isinstance(data['date_of_birth'], str): obj.date_of_birth = datetime.fromisoformat(data['date_of_b irth']).date() return obj </pre>	
<pre> @classmethod def from_row(cls, row: tuple): """ Buat instance dari database row""" if len(row) >= 6: return cls(applicant_id=row[0], first_name=row[1], last_name=row[2], date_of_birth=row[3], address=row[4], phone_number=row[5]) return cls() </pre>	Fungsi ini bertujuan untuk membuat <i>instance</i> dari hasil <i>query database</i> .
<pre> def __init__(self, detail_id: Optional[int] = None, applicant_id: Optional[int] = None, application_role: Optional[str] = None, cv_path: Optional[str] = None): self.detail_id = detail_id self.applicant_id = applicant_id self.application_role = application_role self.cv_path = cv_path </pre>	Fungsi ini bertujuan untuk menginisialisasi objek detail lamaran dari seorang pelamar kerja.
<pre> def to_dict(self): """ Convert ke dictionary untuk JSON""" return { 'detail_id': self.detail_id, </pre>	Fungsi ini bertujuan untuk mengkonversi objek menjadi <i>dictionary</i> .

<pre> 'applicant_id': self.applicant_id, 'application_role': self.application_role, 'cv_path': self.cv_path } </pre>	
<pre> @classmethod def from_dict(cls, data: Dict[str, Any]): """ Buat instance dari dictionary """ obj = cls() obj.detail_id = data.get('detail_id') obj.applicant_id = data.get('applicant_id') obj.application_role = data.get('application_role') obj.cv_path = data.get('cv_path') return obj </pre>	Fungsi ini bertujuan untuk membuat <i>instance</i> ApplicationDetail dari <i>dictionary</i> .
<pre> @classmethod def from_row(cls, row: tuple): """ Buat instance dari database row""" if len(row) >= 4: return cls(detail_id=row[0], applicant_id=row[1], application_role=row[2], cv_path=row[3]) return cls() </pre>	Fungsi ini bertujuan untuk membuat <i>instance</i> dari <i>database row</i> .

operations.py:

Fungsi/Prosedur	Penjelasan
-----------------	------------

```

@staticmethod
def create_applicant(first_name: str,
last_name: str, date_of_birth:
Optional[date] = None, address:
Optional[str] = None, phone_number:
Optional[str] = None) ->
Optional[int]:
    connection =
get_db_connection()
    if not connection:
        return None

    try:
        cursor =
connection.cursor()
        insert_query = """
            INSERT INTO
ApplicantProfile (first_name,
last_name, date_of_birth, address,
phone_number)
            VALUES (%s, %s, %s,
%s, %s)
"""

        cursor.execute(insert_query,
(first_name, last_name, date_of_birth,
address, phone_number))
        connection.commit()
        applicant_id =
cursor.lastrowid
        return applicant_id
    except Error as e:
        connection.rollback()
        print(f"Error creating
applicant: {e}")
        return None
    finally:
        cursor.close()

```

Fungsi ini bertujuan untuk membuat profil pelamar baru di *database*. Fungsi ini menggunakan `cursor.lastrowid()` untuk mendapatkan ID yang baru dibuat.

```

@staticmethod
def create_application(applicant_id:
int, application_role: str, cv_path:
str) -> Optional[int]:
    connection =

```

Fungsi ini bertujuan untuk membuat detail lamaran kerja baru dari seorang pelamar.

```

get_db_connection()
    if not connection:
        return None

    try:
        cursor =
connection.cursor()

        insert_query = """
            INSERT INTO
ApplicationDetail (applicant_id,
application_role, cv_path)
            VALUES (%s, %s, %s)
"""

        cursor.execute(insert_query,
(applicant_id, application_role,
cv_path))
        connection.commit()
        application_id =
cursor.lastrowid # ambil ID untuk
application yang baru dibuat
        return application_id
    except Error as e:
        connection.rollback()
        print(f"Error creating
application: {e}")
        return None
    finally:
        cursor.close()

```

```

@staticmethod
def get_all_applications() ->
List[ApplicationDetail]:
    connection =
get_db_connection()
    if not connection:
        return []

    try:
        cursor =
connection.cursor()
        query = """
            SELECT detail_id,

```

Fungsi ini bertujuan untuk mengambil semua data lamaran dari *database*.

```

applicant_id, application_role,
cv_path
        FROM ApplicationDetail
        ORDER BY detail_id
DESC
"""
    cursor.execute(query)
    rows = cursor.fetchall()

    applications = []
    for row in rows:
        app =
ApplicationDetail.from_row(tuple(row))

        applications.append(app)

    return applications
except Error as e:
    print(f"Error getting
applications: {e}")
    return []
finally:
    cursor.close()

```

```

@staticmethod
def
get_application_by_id(application_id:
int) -> Optional[ApplicationDetail]:
    connection =
get_db_connection()
    if not connection:
        return None

    try:
        cursor =
connection.cursor()
        query = """
            SELECT detail_id,
applicant_id, application_role,
cv_path
            FROM ApplicationDetail
            WHERE detail_id = %s
"""
        cursor.execute(query,

```

Fungsi ini bertujuan untuk mengambil detail pelamar berdasarkan suatu ID tertentu.

```

(application_id,))
    row = cursor.fetchone()
    if row:
        return
ApplicationDetail.from_row(tuple(row))
    return None
except Error as e:
    print(f"Error getting
application: {e}")
    return None
finally:
    cursor.close()

```

```

@staticmethod
def
get_application_by_id(application_id:
int) -> Optional[ApplicationDetail]:
    connection =
get_db_connection()
    if not connection:
        return None

    try:
        cursor =
connection.cursor()
        query = """
            SELECT detail_id,
applicant_id, application_role,
cv_path
            FROM ApplicationDetail
            WHERE detail_id = %s
        """
        cursor.execute(query,
(application_id,))
        row = cursor.fetchone()
        if row:
            return
ApplicationDetail.from_row(tuple(row))
        return None
    except Error as e:
        print(f"Error getting
application: {e}")
        return None
    finally:

```

Fungsi ini bertujuan untuk mengambil detail lamaran berdasarkan ID tertentu.

cursor.close()	
<pre> @staticmethod def get_application_by_cv_path(cv_path: str) -> Optional[ApplicationDetail]: connection = get_db_connection() if not connection: return None cursor = None try: cursor = connection.cursor() query = """ SELECT detail_id, applicant_id, application_role, cv_path FROM ApplicationDetail WHERE cv_path = %s """ cursor.execute(query, (cv_path,)) row = cursor.fetchone() # Fetch all remaining results to clear the cursor cursor.fetchall() if row: return ApplicationDetail(detail_id=row[0], applicant_id=row[1], application_role=row[2], cv_path=row[3]) except Error as e: print(f"Error getting application by cv_path: {e}") return None </pre>	Fungsi ini bertujuan untuk menjadi detail lamaran berdasarkan masukan <i>path file CV</i> .

<pre> finally: if cursor: cursor.close() </pre>	
<pre> @staticmethod def delete_application(application_id: int) -> bool: connection = get_db_connection() if not connection: return False try: cursor = connection.cursor() delete_query = "DELETE FROM ApplicationDetail WHERE detail_id = %s" cursor.execute(delete_query, (application_id,)) connection.commit() return cursor.rowcount > 0 except Error as e: connection.rollback() print(f"Error deleting application: {e}") return False finally: cursor.close() </pre>	Fungsi ini bertujuan untuk menghapus lamaran berdasarkan ID.
<pre> @staticmethod def delete_applicant(applicant_id: int) -> bool: """ Delete applicant and all related applications (CASCADE) """ connection = get_db_connection() if not connection: return False try: cursor = connection.cursor() </pre>	Fungsi ini bertujuan untuk menghapus pelamar dan semua detail lamarannya berdasarkan ID.

```

        delete_query = "DELETE
FROM ApplicantProfile WHERE
applicant_id = %s"

cursor.execute(delete_query,
(applicant_id,))
    connection.commit()
    return cursor.rowcount > 0
except Error as e:
    connection.rollback()
    print(f"Error deleting
applicant: {e}")
    return False
finally:
    cursor.close()

```

```

@staticmethod
def clear_all_data() -> bool:
    """ Clear all data from
database """
    connection =
get_db_connection()
    if not connection:
        return False

    try:
        cursor =
connection.cursor()
        cursor.execute("SET
FOREIGN_KEY_CHECKS = 0")

        # hapus semua data dari
tabel
        cursor.execute("DELETE
FROM ApplicationDetail")
        cursor.execute("DELETE
FROM ApplicantProfile")

        # reset untuk
auto-increment
        cursor.execute("ALTER
TABLE ApplicationDetail AUTO_INCREMENT
= 1")
        cursor.execute("ALTER

```

Fungsi ini bertujuan untuk menghapus semua data dari *database*.

```

TABLE ApplicantProfile AUTO_INCREMENT
= 1")

        # aktifkan kembali foreign
key checks
        cursor.execute("SET
FOREIGN_KEY_CHECKS = 1")

        connection.commit()
        print("All data cleared
successfully")
        return True
    except Error as e:
        connection.rollback()
        print(f"Error clearing
data: {e}")
        return False
    finally:
        cursor.close()

```

encryption.py:

Fungsi/Prosedur	Penjelasan
<pre> def __init__(self): # printable ASCII characters (32-126) self.charset = ''.join(chr(i) for i in range(32, 127)) self.charset_size = len(self.charset) self.shift = 47 </pre>	Fungsi ini bertujuan untuk menginisialisasi <i>charset</i> (karakter ASCII 32-126) dan <i>shift value</i> (47) untuk <i>caesar cipher</i> .
<pre> def _caesar_encrypt(self, text: str) -> str: result = "" for char in text: if char == '-': result += char elif char in self.charset: old_index = self.charset.index(char) </pre>	Fungsi ini melakukan enkripsi teks menggunakan <i>caesar cipher</i> dengan <i>shift</i> sebesar 47.

```

        new_index = (old_index
+ self.shift) % self.charset_size
        result +=
self.charset[new_index]
    else:
        result += char
return result

```

```

def _caesar_decrypt(self, text: str)
-> str:
    result = ""
    for char in text:
        if char == '-':
            result += char
        elif char in self.charset:
            old_index =
self.charset.index(char)
            new_index = (old_index
- self.shift) % self.charset_size
            result +=
self.charset[new_index]
        else:
            result += char
    return result

```

```

def _swap_positions(self, text: str)
-> str:
    if len(text) <= 1:
        return text

    text_list = list(text)
    length = len(text_list)

    for i in range(length // 2):
        left_index = i
        right_index = length - 1 -
i

        text_list[left_index],
text_list[right_index] =
text_list[right_index],
text_list[left_index] # swap

    return ''.join(text_list)

```

Fungsi ini bertujuan untuk mendekripsi teks *caesar cipher* dengan membalik *shift* (mengurangi 47).

Fungsi ini bertujuan untuk menukar posisi karakter dari ujung ke tengah (karakter pertama dengan terakhir, kedua dengan kedua terakhir, dst.).

```

def _encrypt_date(self, date_str: str) -> str:
    if not date_str or '-' not in date_str:
        return date_str
    try:
        parts =
date_str.split('-')
        if len(parts) != 3:
            return date_str

        year, month, day = parts

        # tahun dibalik
        encrypted_year =
year[::-1]

        # bulan dishift 4
        month_int = int(month)

        encrypted_month_int =
((month_int - 1 + 4) % 12) + 1
        encrypted_month =
f"{{encrypted_month_int:02d}"

        # hari dishift 7
        day_int = int(day)
        encrypted_day_int =
((day_int - 1 + 7) % 28) + 1
        encrypted_day =
f"{{encrypted_day_int:02d}"

        return
f"{{encrypted_year}}-{encrypted_month}}-{encrypted_day}"
    except:
        return date_str

```

Fungsi ini bertujuan untuk mengenkripsi khusus untuk tanggal agar dapat tetap dimasukkan ke dalam *database*.

```

def _decrypt_date(self, encrypted_date: str) -> str:
    if not encrypted_date or '-' not in encrypted_date:
        return encrypted_date

```

Fungsi ini bertujuan untuk mendekripsi tanggal dalam *database*.

```

try:
    parts =
encrypted_date.split('-')
    if len(parts) != 3:
        return encrypted_date

    year, month, day = parts

    decrypted_year =
year[::-1]

    month_int = int(month)
    decrypted_month_int =
((month_int - 1 - 4) % 12) + 1
    decrypted_month =
f'{decrypted_month_int:02d}'

    day_int = int(day)
    decrypted_day_int =
((day_int - 1 - 7) % 28) + 1
    decrypted_day =
f'{decrypted_day_int:02d}'

    return
f'{decrypted_year}-{decrypted_month}-{decrypted_day}'
except:
    return encrypted_date

```

```

def encrypt(self, plaintext: str) ->
str:
    """
    1. Apply caesar cipher (shift
47)
    2. Swap character positions
    """
    if not plaintext:
        return ""

    result =
self._caesar_encrypt(plaintext)
    result =
self._swap_positions(result)

```

Fungsi ini bertujuan untuk menggabungkan *caesar cipher* dengan *position swapping*.

<pre> return result </pre>	
<pre> def decrypt(self, ciphertext: str) -> str: """ 1. Reverse character position swapping 2. Reverse caesar cipher """ if not ciphertext: return "" result = self._swap_positions(ciphertext) result = self._caesar_decrypt(result) return result </pre>	Fungsi ini bertujuan untuk membalikkan enkripsi.
<pre> def encrypt_date(self, date_str: str) -> str: """ Special encryption untuk date fields """ if not date_str: return "" return self._encrypt_date(date_str) </pre>	Fungsi ini bertujuan untuk melakukan <i>wrapping</i> dari fungsi <code>_encrypt_date</code> .
<pre> def decrypt_date(self, encrypted_date: str) -> str: """ Special decryption untuk date fields """ if not encrypted_date: return "" return self._decrypt_date(encrypted_date) </pre>	Fungsi ini bertujuan untuk melakukan <i>wrapping</i> dari fungsi <code>_decrypt_date</code> .
<pre> def get_encryption_instance() -> Encryption: global _encryption_instance </pre>	Fungsi ini merupakan <i>singleton pattern</i> yang memastikan bahwa hanya ada satu <i>instance</i> Encryption.

<pre> if _encryption_instance is None: _encryption_instance = Encryption() return _encryption_instance </pre>	
<pre> def encrypt(data: str) -> str: if not data: return "" return get_encryption_instance().encrypt(data)) </pre>	Fungsi ini merupakan fungsi global enkripsi yang menggunakan <i>instance singleton</i> yang telah didefinisikan.
<pre> def decrypt(encrypted_data: str) -> str: if not encrypted_data: return "" return get_encryption_instance().decrypt(encrypted_data) </pre>	Fungsi ini merupakan fungsi global dekripsi menggunakan <i>instance singleton</i> yang telah didefinisikan.
<pre> def encrypt_date(date_str: str) -> str: if not date_str: return "" return get_encryption_instance().encrypt_date(date_str) </pre>	Fungsi ini melakukan fungsi global enkripsi tanggal menggunakan <i>instance singleton</i> yang telah didefinisikan.
<pre> def decrypt_date(encrypted_date: str) -> str: if not encrypted_date: return "" return get_encryption_instance().decrypt_date(encrypted_date) </pre>	Fungsi ini merupakan fungsi global dekripsi tanggal menggunakan <i>instance singleton</i> yang telah didefinisikan.

4.3 Tata Cara Penggunaan Program

Penggunaan Program terbagi menjadi dua bagian. Bagian pertama adalah memasukkan data applicant ke dalam basis data disertai dengan CV bertipe PDF yang sesuai dengan cv_path

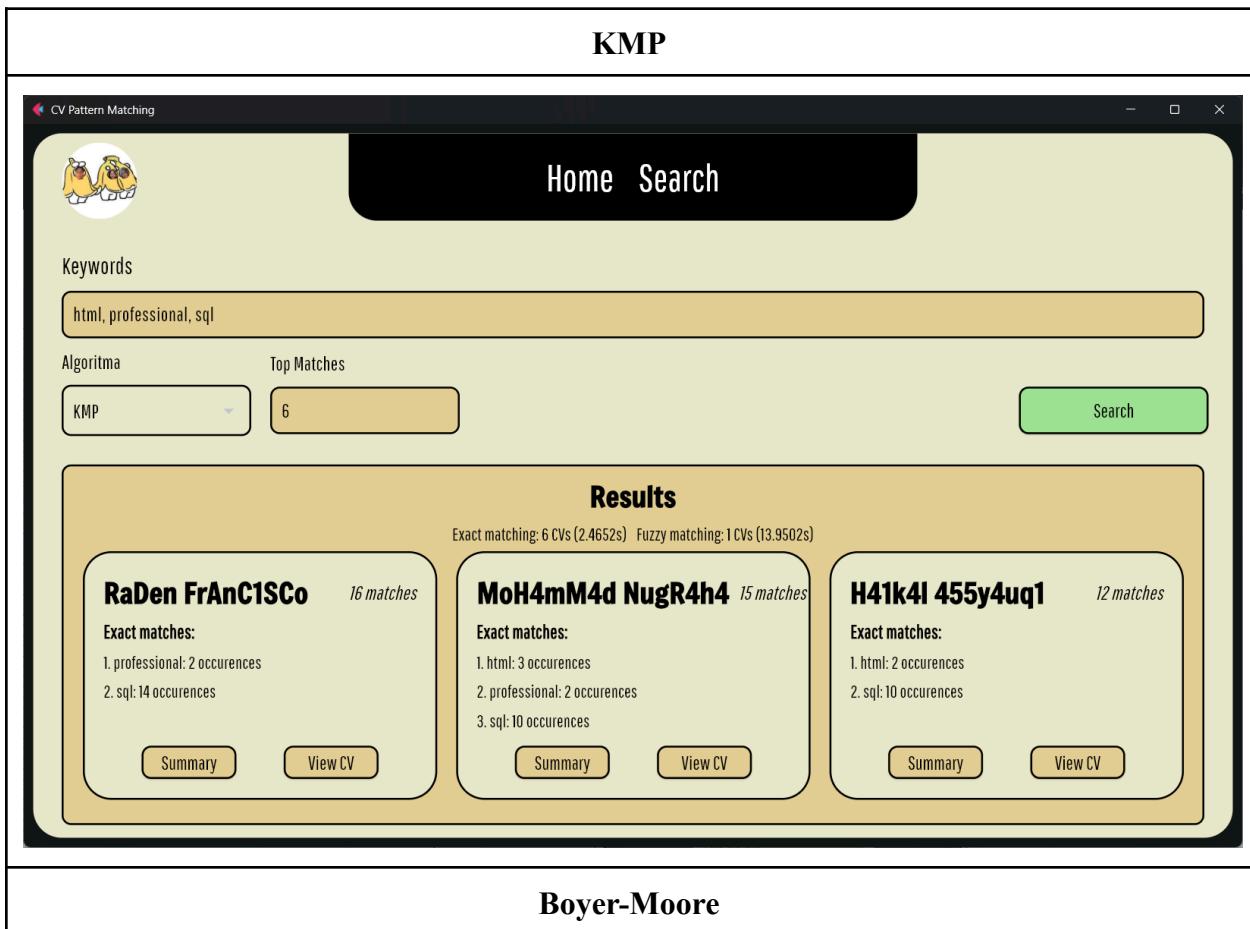
pada basis data. Bagian kedua adalah membuka program melalui Flet. Berikut adalah langkah lengkap untuk penggunaan program.

1. Melakukan git clone untuk repository program.
2. Memberikan *seeding* yang tepat untuk basis data dan memasukkan PDF yang tertera pada basis data.
3. Membuka VS Code atau IDE lainnya dan menjalankan program. Jangan lupa untuk melakukan instalasi pada requirements.txt dengan perintah `pip install -r requirements.txt`
4. Program akan melakukan ekstrasi pada seluruh CV yang disebutkan pada basis data terlebih dahulu sebelum GUI Flet muncul.
5. Setelah ekstraksi selesai, program memutar lagu latar yang dapat dimatikan dengan tombol yang berada di kanan bawah.
6. Pengguna dapat melakukan pencarian dengan pergi ke halaman pencarian (search).
7. Pada halaman pencarian, pengguna diharuskan untuk memasukkan setidaknya satu kata kunci, memilih algoritma pencarian, dan memasukkan jumlah hasil pencarian yang diingkan.
8. Ketika menekan tombol ‘search’, program akan melakukan pencarian dan hasil akan ditampilkan ketika program selesai mencari.
9. Pada setiap CV yang ditampilkan, terdapat pilihan untuk melihat CV asli dan melihat ringkasan dari CV. CV asli dilihat dengan menekan tombol ‘view CV’ dan ringkasan CV diakses dengan menekan tombol ‘summary’

4.4 Hasil Pengujian

4.4.1 Pengujian Biasa

1. Mencari *pattern* html, professional, dan sql untuk 6 buah CV.



CV Pattern Matching

The screenshot shows the 'CV Pattern Matching' application interface. At the top, there's a logo of two cartoonish yellow characters and a navigation bar with 'Home' and 'Search' buttons. Below the navigation is a 'Keywords' input field containing 'html, professional, sql'. Underneath it, there are dropdown menus for 'Algoritma' (set to 'Boyer-Moore') and 'Top Matches' (set to '6'), and a green 'Search' button. The main area is titled 'Results' and displays three search results boxes. Each result box contains the name of the pattern, the number of matches, and a breakdown of exact and fuzzy matches. At the bottom of each result box are 'Summary' and 'View CV' buttons.

Keywords

html, professional, sql

Algoritma Top Matches

Boyer-Moore 6 Search

Results

Exact matching: 6 CVs (1.5493s) Fuzzy matching: 0 CVs (14.3767s)

RaDen FrAnC1SCo 16 matches

Exact matches:

- 1. professional: 2 occurrences
- 2. sql: 14 occurrences

MoH4mM4d NugR4h4 15 matches

Exact matches:

- 1. html: 3 occurrences
- 2. professional: 2 occurrences
- 3. sql: 10 occurrences

H41k4l 455y4uq1 12 matches

Exact matches:

- 1. html: 2 occurrences
- 2. sql: 10 occurrences

Summary View CV

Summary View CV

Summary View CV

Aho-Corasick

CV Pattern Matching

The screenshot shows the 'CV Pattern Matching' application interface, identical to the one above but with a different algorithm selected. The 'Algoritma' dropdown is now set to 'Aho-Corasick'. The search results are displayed in three boxes, each showing a pattern name, the number of matches, and a breakdown of exact and fuzzy matches. The results are: '1khwan 4lh4k1m' (10 matches), '4l4nd MuL14' (10 matches), and 'RaDen FrAnC1SCo' (8 matches). Each result box has 'Summary' and 'View CV' buttons at the bottom.

Keywords

html, professional, sql

Algoritma Top Matches

Aho-Corasick 6 Search

Results

Exact matching: 4 CVs (0.9600s) Fuzzy matching: 5 CVs (18.7696s)

1khwan 4lh4k1m 10 matches

Fuzzy matches:

- 1. professional: 1 occurrence
- 2. sql: 9 occurrences

4l4nd MuL14 10 matches

Exact matches:

- 1. professional: 1 occurrence

Fuzzy matches:

- 1. html: 2 occurrences

RaDen FrAnC1SCo 8 matches

Fuzzy matches:

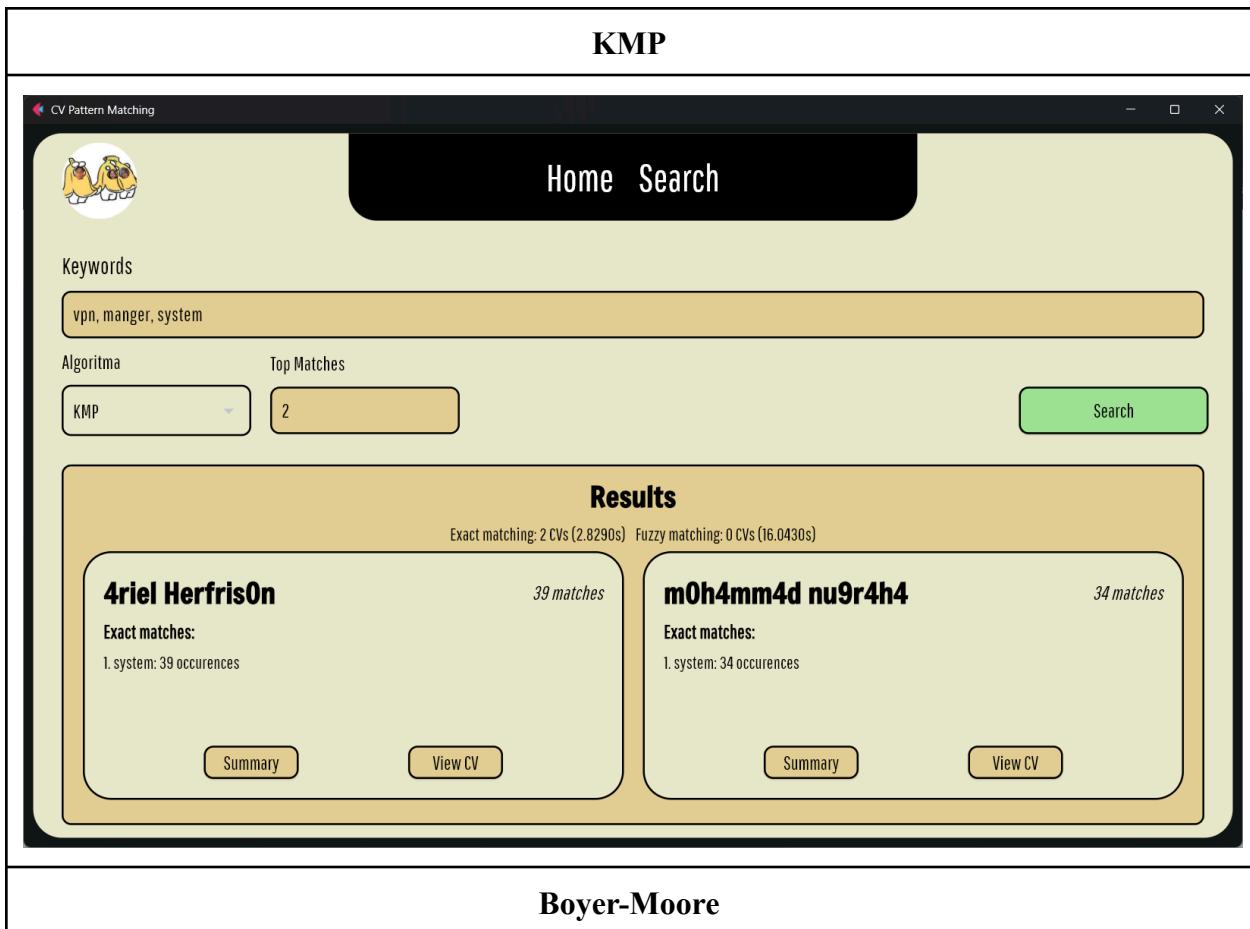
- 1. professional: 2 occurrences
- 2. sql: 6 occurrences

Summary View CV

Summary View CV

Summary View CV

2. Mencari *pattern* vpn, manger (*typo*), dan system untuk 2 buah CV.



CV Pattern Matching

Home Search

Keywords: vpn, manger, system

Algoritma: Boyer-Moore Top Matches: 2

Search

Results

Exact matching: 2 CVs (1.4434s) Fuzzy matching: 1 CVs (15.8117s)

4riel HerfrisOn 39 matches

Exact matches:
1. system: 39 occurrences

Summary View CV

R4d3N Fr4nc15co 34 matches

Exact matches:
1. system: 30 occurrences

Fuzzy matches:
1. manger: 4 occurrences

Summary View CV

Aho-Corasick

CV Pattern Matching

Home Search

Keywords: vpn, manger, system

Algoritma: Aho-Corasick Top Matches: 2

Search

Results

Exact matching: 2 CVs (0.7017s) Fuzzy matching: 1 CVs (16.9385s)

Ikhw4n ALH4KIM 30 matches

Exact matches:
1. system: 5 occurrences

Fuzzy matches:
1. manger: 25 occurrences

Summary View CV

4riel HerfrisOn 27 matches

Exact matches:
1. system: 27 occurrences

Summary View CV

3. Mencari *pattern* colmbia (*typo*) dan technician (*typo*) untuk 5 buah CV

KMP

Keywords

colmbia, technician

Algoritma Top Matches

KMP 5 Search

Results

Exact matching: 0 CVs (2.2005s) Fuzzy matching: 5 CVs (23.2583s)

Result	Matches
AL4ND MUL1A	17 matches
ALaNd MuLIA	15 matches
4riel HerfrisOn	14 matches

Fuzzy matches:
1. technician: 17 occurrences

Fuzzy matches:
1. technician: 15 occurrences

Fuzzy matches:
1. technician: 14 occurrences

Summary View CV

Summary View CV

Summary View CV

Boyer-Moore

CV Pattern Matching

The screenshot shows the 'CV Pattern Matching' application interface. At the top, there is a logo of two cartoonish yellow monkeys and a navigation bar with 'Home' and 'Search' buttons. Below the navigation bar is a 'Keywords' input field containing 'colmbia, technician'. Underneath the input field are dropdown menus for 'Algoritma' (set to 'Boyer-Moore') and 'Top Matches' (set to '5'), and a green 'Search' button. The main area is titled 'Results' and displays three search results. Each result card includes the search term, the number of matches, and a breakdown of exact and fuzzy matches. There are also 'Summary' and 'View CV' buttons for each result.

Keywords

colmbia, technician

Algoritma Top Matches

Boyer-Moore 5 Search

Results

Exact matching: 0 CVs (1.2162s) Fuzzy matching: 5 CVs (22.5484s)

Term	Matches
AL4ND MUL1A	17 matches
ALaNd MuLiA	15 matches
4riel HerfrisOn	14 matches

Fuzzy matches:

- 1. technician: 17 occurrences
- 1. technician: 15 occurrences
- 1. technician: 14 occurrences

Summary View CV

Summary View CV

Summary View CV

Aho-Corasick

CV Pattern Matching

The screenshot shows the 'CV Pattern Matching' application interface, identical to the one above but with a different search algorithm selected. The 'Algoritma' dropdown is now set to 'Aho-Corasick'. The results are identical to the Boyer-Moore search, showing three terms with their respective match counts and fuzzy match details. The 'Summary' and 'View CV' buttons are present for each result.

Keywords

colmbia, technician

Algoritma Top Matches

Aho-Corasick 5 Search

Results

Exact matching: 0 CVs (0.8074s) Fuzzy matching: 5 CVs (23.6900s)

Term	Matches
AL4ND MUL1A	17 matches
ALaNd MuLiA	15 matches
4riel HerfrisOn	14 matches

Fuzzy matches:

- 1. technician: 17 occurrences
- 1. technician: 15 occurrences
- 1. technician: 14 occurrences

Summary View CV

Summary View CV

Summary View CV

4. Mencari *pattern* in-house, inventory, admin, microsoft, dan network untuk 1 buah CV

The screenshot shows the 'CV Pattern Matching' application interface. At the top, there is a navigation bar with icons for Home and Search. Below the navigation bar, there is a section for 'Keywords' containing the text 'in-house, inventory, admin, microsoft, network'. Underneath this, there are dropdown menus for 'Algoritma' (set to 'KMP') and 'Top Matches' (set to '1'). A green 'Search' button is located to the right of these fields. The main area is titled 'Results' and displays the following information:
Exact matching: 1 CVs (4.2933s) Fuzzy matching: 0 CVs (35.6715s)
A result card for '4hmad Rafi' is shown, indicating 47 matches. The card lists exact matches:
1. inventory: 2 occurrences
2. admin: 8 occurrences
3. microsoft: 10 occurrences
At the bottom of the result card are two buttons: 'Summary' and 'View CV'. The overall interface has a light beige background with dark borders around the main sections.

CV Pattern Matching



Home Search

Keywords
in-house, inventory, admin, microsoft, network

Algoritma Top Matches
Boyer-Moore 1 Search

Results
Exact matching: 1 CVs (1.9858s) Fuzzy matching: 0 CVs (41.3546s)

4hmad Rafi 47 matches

Exact matches:
1. inventory: 2 occurrences
2. admin: 8 occurrences
3. microsoft: 10 occurrences

Summary View CV

Aho-Corasick

CV Pattern Matching



Home Search

Keywords
in-house, inventory, admin, microsoft, dan network

Algoritma Top Matches
Aho-Corasick 1 Search

Results
Exact matching: 1 CVs (0.7434s) Fuzzy matching: 1 CVs (39.1690s)

Ahm4d r4f1 31 matches

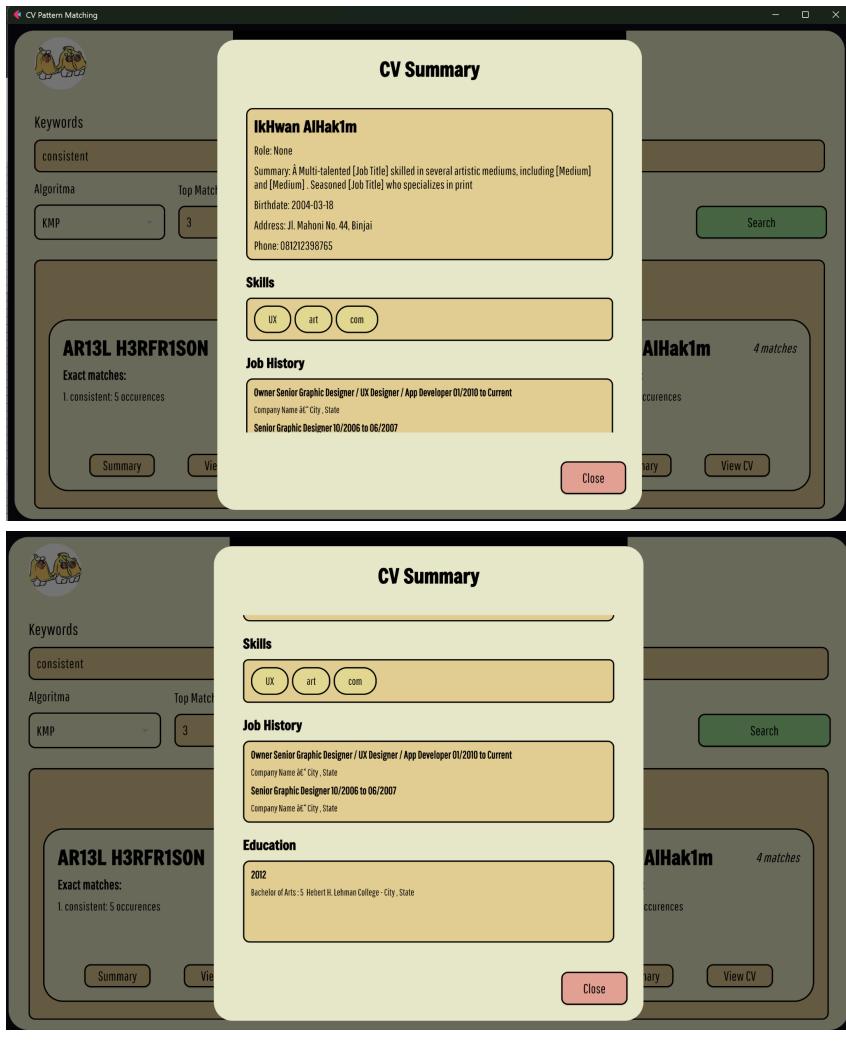
Exact matches:
1. in-house: 2 occurrences

Fuzzy matches:
1. microsoft: 29 occurrences

Summary View CV

4.4.2 Pengujian Lainnya

Membuka ringkasan CV



Membuka CV asli

The screenshot shows a dual-pane interface. On the left, a PDF document titled 'OWNER SENIOR GRAPHIC DESIGNER / UX DESIGNER / APP DEVELOPER' is displayed. It lists various skills and work experience. On the right, a search results page for 'IkHwan AlHakim' is shown, indicating 4 matches found. The search interface includes a 'Search' button and a 'View CV' button.

Kasus tidak ada masukan kata kunci

The screenshot shows a search interface with a 'Keywords' input field containing 'ex React, Express, HTML'. Below it, an 'Algoritma' dropdown is set to 'KMP' and a 'Top Matches' button is visible. A modal window titled 'Invalid Input' displays the message 'Please enter at least one keyword.' with an 'OK' button.

Kasus tidak ada masukan algoritma

The screenshot shows a search interface with an empty 'Keywords' input field. Below it, an 'Algoritma' dropdown is set to 'KMP' and a 'Top Matches' button is visible. A modal window titled 'Invalid Input' displays the message 'Please enter at least one keyword.' with an 'OK' button.

Kasus tidak ada masukan top matches

The screenshot shows a search interface with a dark grey header containing a cartoon dog icon, 'Home' in white, and a 'Search' button. Below the header is a form with a 'Keywords' field containing 'skill'. Underneath are two dropdown menus: 'Algoritma' set to 'Select an algorithm' and 'Top Matches' set to '1'. A yellow 'Invalid Input' dialog box is centered, displaying the message 'Please select an algorithm.' with an 'OK' button at the bottom right.

Kasus top matches bukan angka

The screenshot shows a search interface with a dark grey header containing a cartoon dog icon, 'Home' in white, and a 'Search' button. Below the header is a form with a 'Keywords' field containing 'skill'. Underneath are two dropdown menus: 'Algoritma' set to 'KMP' and 'Top Matches' set to 'h'. A yellow 'Invalid Input' dialog box is centered, displaying the message 'Top Matches must be a number.' with an 'OK' button at the bottom right.

Kasus top matches adalah nol

The screenshot shows a search interface with a dark grey header containing a cartoon dog icon, 'Home' in white, and a 'Search' button. Below the header is a form with a 'Keywords' field containing 'skill'. Underneath are two dropdown menus: 'Algoritma' set to 'KMP' and 'Top Matches' set to '0'. A yellow 'Invalid Input' dialog box is centered, displaying the message 'Please enter a number greater than 0 for Top Matches.' with an 'OK' button at the bottom right.

Kasus tidak ditemukan kata kunci di CV manapun

The screenshot shows a search interface with a yellow header bar. On the left is a small cartoon owl icon. To its right are two buttons: 'Home' and 'Search'. Below the header is a search bar containing the text 'halo ihihihih'. Underneath the search bar are two dropdown menus: 'Algoritma' set to 'KMP' and 'Top Matches' set to '10'. To the right of these is a green 'Search' button. The main area is titled 'Results' and contains the text 'Exact matching: 0 CVs (0.3240s) Fuzzy matching: 0 CVs (4.4986s)'. At the bottom of this area is a small illustration of three black, spiky characters. Below the illustration, the text 'No matches found' is displayed.

4.5 Analisis Hasil Pengujian

Pengujian sistem dilakukan untuk mengevaluasi dan membandingkan kinerja dari tiga algoritma pattern matching yang diimplementasikan, yakni *Knuth-Morris-Pratt* (KMP), *Boyer-Moore* (BM), dan *Aho-Corasick*, serta algoritma *Levenshtein Distance* untuk fungsionalitas *fuzzy matching*. Skenario pertama berfokus pada pencarian multi-kata kunci standar (html, professional, sql) untuk mengukur kecepatan *exact match*. Skenario kedua dirancang untuk menguji penanganan *typographical error* dengan memasukkan kata kunci manger (typo dari "manager") untuk memicu mekanisme *fuzzy matching*.

Hasil pengujian kuantitatif menunjukkan perbedaan kinerja yang jelas pada pencarian exact matching. Pada skenario pertama, algoritma *Aho-Corasick* terbukti menjadi yang tercepat dengan waktu eksekusi sekitar 0.96 detik. Kinerja ini secara signifikan melampaui *Boyer-Moore* dengan waktu 1.5493 detik dan *Knuth-Morris-Pratt* dengan 2.4652 detik. Keunggulan *Aho-Corasick* ini konsisten dengan teorinya yang dirancang untuk *multi-pattern matching* secara efisien, memproses semua kata kunci dalam satu kali traversal teks.

Fungsionalitas *fuzzy matching* diuji secara efektif pada skenario kedua saat sistem tidak menemukan kecocokan pasti untuk kata kunci "manger". Sesuai dengan spesifikasi, ketika *exact match* gagal, sistem secara otomatis mengaktifkan pencarian kemiripan menggunakan algoritma *Levenshtein Distance*. Meskipun fitur ini berhasil menemukan kandidat yang relevan dengan

mencocokkannya ke “manager”, hasil pengujian menunjukkan adanya biaya komputasi yang signifikan. Sebagai contoh, pada algoritma *Boyer-Moore* di skenario kedua, waktu pencarian berlangsung sekitar 16 detik khusus untuk proses *fuzzy matching*. Penambahan waktu ini menjadi *trade-off* antara kecepatan dan fleksibilitas pencarian, namun fungsionalitas ini membantu meningkatkan pengalaman pengguna dengan mengakomodasi kemungkinan kesalahan *input* atau variasi penulisan.

BAB 5

KESIMPULAN, SARAN, DAN REFLEKSI

5.1 Kesimpulan

Tugas Besar 3 IF2211 Strategi Algoritma berkaitan dengan pemanfaatan *pattern matching* untuk membangun sistem ATS (*Application Tracking System*) berbasis CV Digital telah diselesaikan oleh tim pengembang. Sistem yang telah dibangun ini mampu melakukan fungsi utama sebagai *Applicant Tracking System* untuk menyaring dan mencocokkan informasi kandidat dari berkas lamaran yang berformat PDF. Aplikasi yang dikembangkan mengimplementasikan ekstraksi teks dari dokumen PDF menggunakan *Regular Expression* untuk mendapatkan data-data penting secara otomatis. Kemudian, diterapkan juga tiga algoritma *pattern matching*, yakni Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick. Selain itu, algoritma Levenshtein Distance juga diterapkan untuk menangani *fuzzy matching*. Seluruh profil pelamar dan hasil ekstraksi berhasil disimpan dan dikelola dalam basis data MySQL.

Dari hasil pengujian performa yang dilakukan terhadap ketiga algoritma *exact matching*, algoritma Aho-Corasick terbukti jauh lebih unggul dan efisien karena hanya membutuhkan satu kali proses traversal teks untuk mencocokkan semua kata kunci secara simultan. Namun, algoritma lain tetap dapat memberikan hasil pencocokan dalam waktu yang memadai, khususnya sebagai dampak dari penggunaan *multi-threading* yang memangkas waktu berjalannya algoritma untuk banyak *pattern* dan banyak *teks* sekaligus. Penggunaan algoritma *Levenshtein Distance* memberikan pendekatan yang baik dalam mengatasi *typo* pada masukan pengguna dan melakukan *fuzzy matching*. *Multi-threading* turut memiliki peran yang krusial terhadap *fuzzy matching* seperti dalam melakukan *exact matching*.

Segala macam bentuk pengerjaan, seperti teori yang digunakan, hasil pembentukan source code, serta implementasi fungsi telah kelompok dokumentasikan secara terstruktur melalui laporan ini. Walaupun masih jauh dari kata sempurna, dengan berbangga hati kelompok

telah berhasil membentuk sistem ATS (*Application Tracking System*) berbasis CV Digital yang dapat dijalankan untuk melakukan pencocokan dan pencarian informasi pelamar kerja sebagai hasil akhir dari tugas besar ketiga mata kuliah IF2211 Strategi Algoritma.

5.2 Saran

1. Untuk penurunan Tugas Besar dilain kesempatan, apabila memungkinkan, lebih baik untuk tidak dilakukan saat masa ujian. Plis kak ini 2 dari 3 anggota besok uas aep ga ada yang sempet belajar 😅

5.3 Refleksi

Pengerjaan tugas besar ini memberikan pengalaman belajar bagi tim pengembang, khususnya dalam menerapkan berbagai strategi algoritma pada sebuah masalah di dunia nyata. Mengintegrasikan semua komponen ini, seperti GUI, logika algoritma yang kompleks, dan basis data menjadi satu aplikasi yang utuh merupakan latihan rekayasa perangkat lunak yang mengajarkan tim pengembang terkait pentingnya desain yang modular dan bersih. Akhir kata,

Bertha: ehehehe balik ke Flet lagi

Max: makasih stima mamah saya seneng dikasih coklat 🤗

Grace: seru tapi stress banget kak T_T

LAMPIRAN

Tautan Repository Github

https://github.com/BerthaSoliany/Tubes3_ikandanpisang

Tautan Video

<https://www.youtube.com/watch?v=FV5EBIXQrR0>

Hasil Akhir Tugas Besar

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 Levenshtein Distance dapat mengukur kemiripan kata kunci dengan benar.	✓	
6	Aplikasi dapat menampilkan <i>summary CV applicant</i> .	✓	
7	Aplikasi dapat menampilkan <i>CV applicant</i> secara keseluruhan.	✓	
8	Membuat laporan sesuai dengan spesifikasi.	✓	
9	Membuat bonus enkripsi data profil <i>applicant</i> .	✓	
10	Membuat bonus algoritma Aho-Corasick.	✓	
11	Membuat bonus video dan diunggah pada Youtube.	✓	

DAFTAR PUSTAKA

Aho-Corasick Algorithm (dari CP-Algorithms) CP-Algorithms. Aho-Corasick algorithm. Diakses pada 26 Mei 2025, dari https://cp-algorithms.com/string/aho_corasick.html

ComputerBread. (2024). Aho-Corasick Algorithm - JavaScript Implementation (pt. 2). Diakses pada 26 Mei 2025, dari <https://www.youtube.com/watch?v=jsgLCvOW6Vo>

Munir, Rinaldi. (2025). Pencocokan String (*String/Pattern Matching*). Diakses pada 13 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)

Munir, Rinaldi. (2025). String Matching dengan Regular Expression. Diakses pada 13 Juni 2025, dari
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf)

NeetCode. (2021). Edit Distance - Dynamic Programming - Leetcode 72 - Python. Diakses pada 28 Mei 2025, dari <https://www.youtube.com/watch?v=XYi2-LPrwm4>