LAPORAN TUGAS BESAR 3

IF2211 STRATEGI ALGORITMA

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

Dipersiapkan oleh:

William Gerald Briandelo	13222061
Irgiansyah Mondo	13521167
Azadi Azhrah	12823024

Kelompok atsTracker

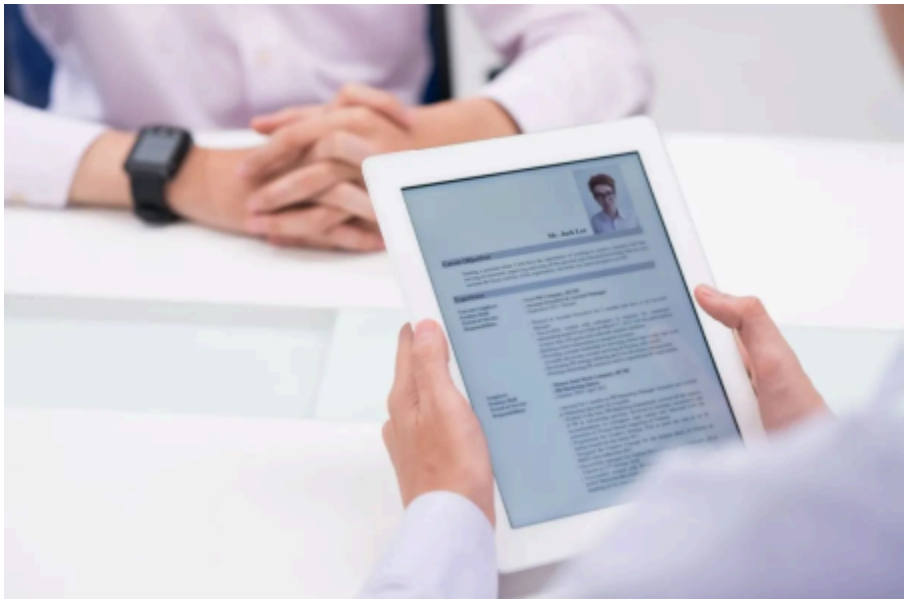
PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

BAB I
DESKRIPSI TUGAS



Gambar 1.1. CV ATS dalam Dunia Kerja
(sumber: https://www.antaranews.com/)

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

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

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

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

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

BAB II
LANDASAN TEORI

2.1. Algoritma KMP

Algoritma Knuth-Morris-Pratt merupakan sebuah algoritma pencarian string yang efisien. Tujuannya adalah untuk menemukan semua kemunculan sebuah pattern di dalam sebuah text. Keunggulan utama KMP dibandingkan dengan brute-force adalah kemampuannya untuk menghindari perbandingan karakter yang berulang. Pada Brute Force, ketika menemukan ketidakcocokan, akan menggeser pola hanya satu karakter ke kanan dan memulai perbandingan dari awal pattern lagi. Hal ini tidak efisien karena mengabaikan informasi yang sudah didapat dari perbandingan sebelumnya. Algoritma KMP mengatasi kelemahan ini dengan melakukan pergeseran yang lebih cerdas berdasarkan struktur internal pattern itu sendiri.

Inti dari mekanisme KMP terletak pada tahap pra-pemrosesan pola untuk membangun sebuah tabel bantu yang disebut Border Function/LPS (Longest Proper Prefix). Sebuah "proper prefix" adalah semua awalan dari sebuah string, kecuali string itu sendiri. Tabel LPS ini menyimpan panjang dari prefiks sejati terpanjang yang juga merupakan sufiks untuk setiap sub-pattern.

Saat proses pencarian berlangsung dan terjadi ketidakcocokan antara karakter di teks dan karakter di pattern pada indeks j, algoritma KMP tidak langsung menggeser pola sejauh satu langkah. Sebaliknya, ia menggunakan tabel LPS. Nilai pada LPS[j-1] memberitahu algoritma berapa banyak karakter pada pattern yang tidak perlu dicocokkan ulang karena karakter tersebut sudah pasti cocok berdasarkan informasi prefiks-sufiks yang telah dihitung. Dengan kata lain, algoritma menggeser pola ke posisi berikutnya yang paling mungkin cocok, tanpa memundurkan penunjuk pada teks. Hal ini secara drastis mengurangi jumlah total perbandingan dan menjamin kinerja dalam waktu linier, yaitu O(n+m), di mana n adalah panjang teks dan m adalah panjang pola.

2.2. Algoritma BM

Algoritma Boyer-Moore (BM) adalah sebuah metode pencocokan pattern yang didasarkan pada dua teknik utama. Pertama adalah teknik looking-glass, di mana pattern (P) dicocokkan dengan teks (T) dengan cara bergerak mundur melalui pattern, dimulai dari karakter terakhirnya. Kedua adalah teknik character-jump, yang digunakan saat terjadi ketidakcocokan (mismatch) antara karakter di teks dengan karakter di pattern.

Saat terjadi ketidakcocokan antara karakter teks T[i] dengan P[j], teknik character-jump akan menentukan seberapa jauh pattern harus digeser ke kanan. Ada tiga kemungkinan kasus yang dipertimbangkan:

- Kasus 1: Jika karakter T[i] yang tidak cocok itu ada di bagian lain dari pattern P, maka P digeser ke kanan untuk menyejajarkan kemunculan terakhir karakter tersebut di P dengan T[i].
- Kasus 2: Jika karakter T[i] ada di dalam pattern tetapi pergeseran ke kanan tidak memungkinkan (karena posisinya ada di sebelah kanan P[j]), maka pattern hanya digeser satu karakter ke kanan.
- Kasus 3: Jika karakter T[i] sama sekali tidak ada di dalam pattern, maka pattern dapat digeser sepenuhnya melewati posisi T[i+1].

Untuk mengimplementasikan pergeseran ini secara efisien, algoritma Boyer-Moore melakukan pra-pemrosesan terhadap pattern dan alfabet yang digunakan untuk membangun fungsi kemunculan terakhir (L()). Fungsi ini memetakan setiap karakter dalam alfabet ke sebuah integer. Secara spesifik, L(x) didefinisikan sebagai indeks terbesar i di mana P[i] == x. Jika karakter x tidak ada dalam pattern, nilainya adalah -1. Informasi ini digunakan untuk menghitung pergeseran secara cepat saat terjadi ketidakcocokan.

2.3. Regular Expression

Regular Expression (Regex) adalah sebuah sekuens karakter yang mendefinisikan sebuah pola pencarian (search pattern). Landasan teorinya tidak berasal dari pemrograman praktis, melainkan dari bidang ilmu komputer teoretis, khususnya teori bahasa formal yang pertama kali diperkenalkan oleh matematikawan Stephen Cole Kleene. Dalam teori ini, Regex adalah notasi standar untuk mendefinisikan sebuah "regular language", yaitu himpunan string yang mematuhi aturan-aturan spesifik. Dengan kata lain, setiap pola Regex secara formal mendeskripsikan sebuah bahasa yang terdiri dari semua string yang mungkin cocok dengan pola tersebut.

Secara konseptual, setiap pola Regex dapat diubah menjadi sebuah mesin abstrak yang disebut Finite Automaton. Mesin ini memiliki serangkaian states dan transisi antar keadaan yang dipicu oleh karakter input. Ketika sebuah mesin Regex mencocokkan sebuah pola dengan teks, ia secara efektif mensimulasikan NFA ini. Proses dimulai dari keadaan awal, dan untuk setiap karakter dalam teks, mesin berpindah ke keadaan berikutnya sesuai dengan aturan transisi. Jika mesin berhasil mencapai accepting state setelah memproses sebagian atau seluruh teks, maka teks tersebut dianggap cocok dengan pola Regex.

2.6. Pembangunan Aplikasi Web

Pada proyek ini, pembangunan aplikasi web dilakukan untuk mempermudah interaksi antara pengguna (rekruter) dengan sistem ATS. Web application dibangun menggunakan pendekatan client-server di mana frontend berfungsi sebagai antarmuka pengguna, sementara backend bertugas memproses data dan menjalankan algoritma pattern matching (KMP dan BM) terhadap CV digital.

Aplikasi ini mendukung pencarian berdasarkan kata kunci dengan metode pattern matching pilihan pengguna dan menampilkan hasil berupa daftar CV yang relevan beserta informasi penting yang diekstraksi dari CV tersebut. Backend terhubung dengan database CV digital dan mengimplementasikan logika algoritma dalam bentuk service API.

2.6.1. Frontend

Pada bagian front end, tugas utamanya adalah untuk berinteraksi secara langsung dengan user melalui GUI dan menerima input dari user. GUI berperan untuk menerima input user berupa algoritma string matching dan menerima input keywords yang dicari dari CV. Selanjutnya user dapat melihat informasi CV yang sesuai dan program dapat menampilkan CV yang direferensikan.

2.6.2. Backend

Pada bagian backend, tugas utamanya adalah untuk memproses data yang diterima dari frontend dan menjalankan algoritma pattern matching (seperti KMP dan Boyer-Moore) terhadap CV digital yang diunggah oleh pelamar kerja. Backend juga bertanggung jawab untuk mengelola database yang menyimpan profil pelamar dan hasil ekstraksi informasi dari CV.

Proses Backend:

1. **Pengolahan Input Pengguna**

   Pengguna akan memilih metode pencocokan pola (KMP atau Boyer-Moore) dan memasukkan kata kunci pencarian. Backend akan menerima input tersebut dan mengeksekusi algoritma yang dipilih untuk mencocokkan kata kunci dengan data dalam CV yang ada di database.

2. **Ekstraksi Data dari CV**

   Backend bertugas untuk mengekstrak teks dari file CV yang ada (baik dalam format PDF atau lainnya) menggunakan pustaka eksternal seperti PyMuPDF. Setelah teks diekstraksi, sistem akan menggunakan algoritma pattern matching (KMP atau Boyer-Moore) untuk mencari pola-pola yang relevan, seperti nama, alamat email, pengalaman kerja, keahlian, dan riwayat pendidikan.

3. **Pencocokan Pola**

   Backend akan menjalankan algoritma pencocokan pola berdasarkan metode yang dipilih oleh pengguna (KMP atau Boyer-Moore). Proses ini dilakukan dengan mencari kata kunci yang diberikan oleh pengguna di dalam teks CV yang telah diekstraksi.

4. **Pengelolaan Hasil Pencarian**

   Setelah pencocokan selesai, backend akan mengumpulkan hasilnya dan

menampilkan daftar CV yang relevan beserta informasi yang diekstraksi dari CV tersebut, seperti nama, email, pengalaman kerja, dan keahlian. Hasil ini akan dikirimkan kembali ke frontend untuk ditampilkan kepada pengguna.

5. **Integrasi dengan Database**

Backend akan terhubung dengan database yang menyimpan data pelamar (seperti nama, email, pengalaman kerja, dll). Data ini akan dikelola dengan menggunakan struktur tabel relasional yang mencakup tabel-tabel seperti ApplicantProfile, ApplicationDetail, dan SearchKeywords.

- **ApplicantProfile** menyimpan data pelamar.
- **ApplicationDetail** mencatat status dan detail lamaran pekerjaan.
- **SearchKeywords** digunakan untuk melacak kata kunci pencarian yang digunakan.

6. **API Endpoint**

Backend akan menyediakan API yang dapat digunakan oleh frontend untuk berkomunikasi dengan sistem. API ini akan menerima permintaan pencarian dan memberikan respons yang berisi hasil pencocokan dan informasi terkait.

Struktur Data di Backend:

Tabel ApplicantProfile menyimpan informasi pelamar, seperti nama depan, nama belakang, nomor telepon, email, dan riwayat pekerjaan. Tabel ApplicationDetail digunakan untuk menyimpan status lamaran pekerjaan yang diajukan oleh pelamar. Sedangkan tabel SearchKeywords akan menyimpan kata kunci yang digunakan untuk pencarian dalam sistem, serta menghitung frekuensi pencariannya.

BAB III
ANALISIS PEMECAHAN MASALAH

3.1. Langkah-Langkah Pemecahan Masalah

Dalam program atsTracker, pertama dikumpulkan kumpulan data CV sebagai database untuk program. Lalu dirancang sebuah database yang menyimpan profil dan detail dari CV. Selanjutnya dari database CV tersebut akan dilakukan proses pencarian sehingga ditemukan CV dengan kriteria sesuai keinginan user dan menampilkan informasi penting terkait CV tersebut.

1. Memproses masukan pengguna

   Program menerima input dari user berupa metode pattern matching KMP atau BM, dan kata kunci dalam pencarian CV.

2. Implementasi algoritma pencarian

   Setelah menerima input tersebut, program akan mencari kata kunci yang diinput user berdasarkan metode pattern matching yang dipilih. Selanjutnya, program akan menampilkan list CV yang paling sesuai dengan kata kunci.

3. Mengambil dan memproses data.

   Setelah itu, pengguna dapat memilih CV dari list tersebut dan ketika dipilih akan menampilkan informasi penting terkait pelamar dan juga CVnya.

4. Menampilkan data secara full

   Selanjutnya, pengguna dapat memilih untuk melihat file CV secara langsung, dan program akan mengantarkan pengguna ke file CV yang tersimpan.

3.2. Proses Pemetaan Masalah dengan algoritma KMP dan BM

Untuk memecahkan masalah pencarian informasi dalam CV digital, dilakukan pemetaan sebagai berikut:

a. Format Dokumen CV

Dokumen CV bisa dalam berbagai format (PDF atau teks mentah). Oleh karena itu, proses awal adalah mengekstrak isi teks dari CV menggunakan pustaka eksternal (PyMuPDF berbasis Python). Hasil ekstraksi berupa string panjang yang siap diproses algoritma pattern matching.

b. Ekstraksi Informasi

Pattern yang dicari dalam teks CV meliputi: nama, email, pengalaman kerja, keahlian, dan pendidikan. Setiap pattern dicari menggunakan algoritma KMP atau BM yang dipilih pengguna.

c. Pencocokan dan Penyimpanan

Hasil pencarian pattern yang cocok ditandai, disimpan dalam struktur data, dan ditampilkan kepada pengguna dalam bentuk tabel atau highlight.

3.1.1. KMP

1. Identifikasi Input
   Tentukan string utama yang akan menjadi target pencarian dan string pola yang ingin ditemukan. Algoritma ini ideal untuk mencari satu pola spesifik di dalam sebuah teks.

2. Preprocessing Data
   Analisis Pola untuk membuat sebuah array LPS. Array ini menjadi kunci algoritma yang menyimpan informasi pergeseran cerdas berdasarkan struktur internal dari Pola itu sendiri.

3. Eksekusi Pencarian
   Lakukan pencocokan dari kiri ke kanan. Ketika terjadi ketidakcocokan, gunakan array LPS untuk menggeser Pola secara efisien tanpa perlu memundurkan pointer pada Teks.

3.1.2. BM

1. Identifikasi Input
   Sama seperti KMP, tentukan Teks sebagai sumber pencarian dan Pola sebagai kata kunci. Boyer-Moore sangat unggul pada teks umum dengan pola yang relatif panjang.

2. Preprocessing Data
   Buat Tabel Last Occurrence dari Pola. Tabel ini mencatat posisi terakhir setiap karakter, yang menjadi dasar untuk melakukan lompatan jauh.

3. Eksekusi Pencarian
   Lakukan pencocokan dari kanan ke kiri. Saat terjadi ketidakcocokan, gunakan Tabel Last Occurrence untuk menghitung dan melakukan pergeseran Pola secara signifikan, seringkali melewati beberapa karakter sekaligus

3.3. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

Fitur Fungsional Aplikasi Web:

- Input kata kunci pencarian dan pemilihan algoritma.
- Proses pencocokan pattern dalam teks CV.
- Penilaian kesesuaian CV berdasarkan kemunculan kata kunci.
- Visualisasi hasil pencarian.
- Navigasi ke file CV asli.

Arsitektur Aplikasi Web

- **Client (Frontend):** Menyediakan form dan tampilan hasil.
- **Server (Backend):** Menerima input, melakukan pattern matching, dan mengirimkan hasil kembali.
- **Database:** Menyimpan data CV dan hasil ekstraksi.

3.4. Contoh Ilustrasi Kasus

3.4.1. KMP

Teks (T): ABABDABABC

Pola (P): ABABC

Pertama, buat array LPS untuk pola P = "ABABC". Array ini memberitahu panjang prefix terpanjang yang juga merupakan akhiran untuk setiap sub-pola.

P[0] = A → LPS = 0

P[0..1] = AB → (Awalan: A; Akhiran: B) → LPS = 0

P[0..2] = ABA → (Awalan: A, AB; Akhiran: A, BA) → Cocok "A" → LPS = 1

P[0..3] = ABAB → (Awalan: A, AB, ABA; Akhiran: B, AB, BAB) → Cocok "AB" → LPS = 2

P[0..4] = ABABC → Tidak ada awalan yang cocok dengan akhiran → LPS = 0

Hasil akhir Array LPS untuk ABABC adalah: [0, 0, 1, 2, 0].

Proses Pencarian:

i=0, j=0

T: ABABDABABC

P: ABABC

Tidak cocok pada j=4

Karena tidak cocok pada j=4, dan lps[j-1] = lps[3] = 2. Geser pola sehingga awalan sepanjang 2 karakter (AB) dari pola selaras dengan teks. Pointer j diatur ke 2. Pointer i tetap di posisi 4.

Sekarang bandingkan T[4] dengan P[2] → (D != A).

Ketidakcocokan terjadi pada j=2. Dan lps[j-1] = lps[1] = 0. Maka, pointer j diatur ke 0

Bandingkan T[4] dengan P[0] → (D != A). Karena masih tidak cocok dan j sudah 0, geser pointer teks i ke posisi berikutnya (i=5).

Selanjutnya lakukan matching dengan i=5 dan j=0, dan ditemukan bahwa pattern sudah cocok.

3.4.2. BM

Teks (T): ABABC

Pola (P): ABC

Buat tabel Last Occurance yang menyimpan kemunculan karakter dari karakter yang match, dan diperoleh tabel sebagai berikut { 'A': 0, 'B': 1, 'C': 2 }.

Bandingkan T[2] dengan P[2] → (A != C). Ketidakcocokan. Karakter yang tidak cocok pada teks adalah 'A'.

Lalu geser 2 langkah ke kanan, dan melalui matching berikutnya, sudah diperoleh hasil yang cocok.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1. Spesifikasi Teknis Program

4.1.1. Struktur Data

```
CREATE TABLE IF NOT EXISTS ApplicantProfile (
    applicant_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100),
    phone_number VARCHAR(20),
    email VARCHAR(150) UNIQUE,
    address TEXT,
    date_of_birth DATE,
    summary TEXT,
    skills TEXT,
    experience TEXT,
    education TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

Tabel ini dirancang untuk menyimpan data lengkap dari para pelamar kerja. Setiap baris dalam tabel mewakili satu pelamar unik, yang diidentifikasi oleh applicant_id yang akan terisi secara otomatis dan selalu unik untuk setiap pelamar baru. Informasi yang disimpan mencakup data pribadi seperti nama, kontak (telepon dan email), alamat, tanggal lahir, serta data profesional seperti ringkasan diri, keahlian (skills), pengalaman kerja, dan riwayat pendidikan.

Fitur penting dari tabel ini adalah adanya kolom email yang diatur sebagai UNIQUE, yang memastikan tidak ada dua pelamar yang dapat mendaftar dengan alamat email yang sama. Selain itu, tabel ini dilengkapi dengan dua kolom timestamp otomatis: created_at akan mencatat waktu saat data pelamar pertama kali dibuat, dan updated_at akan secara otomatis memperbarui waktunya setiap kali ada perubahan pada data pelamar tersebut. Ini sangat berguna untuk melacak histori dan integritas data. Video Deep Research Canvas

```
CREATE TABLE IF NOT EXISTS ApplicationDetail (
    application_id INT AUTO_INCREMENT PRIMARY KEY,
    applicant_id INT NOT NULL,
    application_role VARCHAR(100) NOT NULL,
    cv_path VARCHAR(500) NOT NULL,
    application_status ENUM('pending', 'reviewed', 'shortlisted',
'rejected') DEFAULT 'pending',
    applied_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    notes TEXT,
    FOREIGN KEY (applicant_id) REFERENCES ApplicantProfile(applicant_id)
ON DELETE CASCADE
);
```

Tabel ApplicationDetail berfungsi untuk mencatat setiap detail dari lamaran pekerjaan yang masuk. Setiap baris dalam tabel ini merepresentasikan satu lamaran spesifik, yang diidentifikasi oleh application_id. Kolom applicant_id bertindak sebagai penghubung (Foreign Key) ke tabel ApplicantProfile, memastikan bahwa setiap lamaran terikat pada seorang pelamar yang ada. Tabel ini menyimpan informasi penting seperti posisi yang dilamar (application_role), lokasi penyimpanan CV (cv_path), dan catatan tambahan dari perekrut.

Fitur utama dari tabel ini adalah kolom application_status yang menggunakan tipe data ENUM, yang membatasi status lamaran hanya pada nilai-nilai yang telah ditentukan ('pending', 'reviewed', 'shortlisted', 'rejected'), sehingga menjaga konsistensi data. Selain itu, klausa ON DELETE CASCADE pada Foreign Key adalah mekanisme pengaman yang sangat penting; jika sebuah data pelamar dihapus dari tabel ApplicantProfile, maka semua data lamaran yang terkait dengan pelamar tersebut di tabel ini akan ikut terhapus secara otomatis. Ini mencegah adanya "data yatim" dan menjaga integritas database.

```
CREATE TABLE IF NOT EXISTS SearchKeywords (
    keyword_id INT AUTO_INCREMENT PRIMARY KEY,
    keyword VARCHAR(100) NOT NULL,
    search_count INT DEFAULT 1,
    last_searched TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE KEY unique_keyword (keyword)
);
```

SearchKeywords dirancang untuk melacak dan menganalisis kata kunci pencarian yang digunakan dalam sebuah sistem. Setiap baris dalam tabel ini mewakili satu kata kunci unik. Tabel ini berfungsi untuk mencatat kata kunci itu sendiri (keyword), menghitung frekuensi pencariannya (search_count), dan merekam kapan terakhir kali kata kunci tersebut dicari (last_searched).

Fitur yang paling penting dari tabel ini adalah UNIQUE KEY pada kolom keyword. Kendala ini memastikan bahwa tidak ada kata kunci yang sama bisa tersimpan lebih dari satu kali. Dengan demikian, ketika seorang pengguna mencari kata kunci yang sudah ada, sistem tidak akan membuat baris baru, melainkan cukup memperbarui search_count dan last_searched pada baris yang sudah ada. Ini membuat tabel ini sangat efisien untuk mengukur popularitas dan relevansi kata kunci dari waktu ke waktu.

4.1.2. Fungsi dan Prosedur

```
def boyer_moore(text, pattern)
```

Fungsi ini menerapkan algoritma Boyer-Moore untuk menemukan posisi kemunculan pertama sebuah pattern di dalam text. Cara kerjanya yang unik adalah dengan melakukan pencocokan dari kanan ke kiri. Sebelum mencari, fungsi ini membuat tabel bad_char yang mencatat posisi terakhir setiap karakter di dalam pattern. Ketika terjadi ketidakcocokan antara karakter di text dengan di pattern, tabel bad_char ini digunakan untuk menentukan "lompatan" atau pergeseran pattern ke kanan sejauh mungkin. Teknik ini membuat algoritma Boyer-Moore sangat efisien karena dapat melewati sebagian besar teks tanpa perlu membandingkan setiap karakter. Fungsi akan mengembalikan indeks awal dari pattern yang ditemukan pertama kali atau -1 jika tidak ditemukan.

```
def boyer_moore_all(text, pattern)
```

Fungsi ini adalah variasi dari algoritma Boyer-Moore yang dirancang untuk menemukan semua posisi kemunculan pattern di dalam text. Sama seperti fungsi sebelumnya, ia menggunakan tabel bad_char dan metode perbandingan dari kanan ke kiri untuk efisiensi. Bedanya, ketika sebuah kecocokan penuh ditemukan, fungsi ini akan menyimpan posisi tersebut dan kemudian menghitung pergeseran berikutnya untuk melanjutkan pencarian di sisa teks. Perhitungan pergeseran setelah menemukan kecocokan diatur sedemikian rupa agar tidak melewatkan kemungkinan adanya pattern lain yang tumpang tindih. Pada akhirnya, fungsi ini akan mengembalikan sebuah daftar yang berisi semua indeks awal di mana pattern ditemukan.

```
def kmp_search(text, pattern)
```

Fungsi ini mengimplementasikan algoritma Knuth-Morris-Pratt untuk menemukan posisi kemunculan pertama dari sebuah pattern di dalam text. Awalnya, fungsi ini memanggil compute_lps untuk membuat tabel lps yang berfungsi sebagai panduan untuk menggeser pattern secara efisien saat terjadi ketidakcocokan. Selama fase pencocokan, fungsi ini membandingkan karakter dari text dan pattern. Jika terjadi ketidakcocokan, tabel lps digunakan untuk menentukan seberapa jauh pattern bisa digeser tanpa perlu mengulang perbandingan karakter yang sudah cocok, sehingga proses pencarian menjadi lebih cepat. Fungsi akan mengembalikan indeks awal dari pattern yang ditemukan atau -1 jika tidak ada yang cocok.

```
def kmp_search_all(text, pattern)
```

Fungsi ini merupakan modifikasi dari algoritma KMP yang bertujuan untuk menemukan semua posisi kemunculan dari pattern di dalam text. Prosesnya mirip dengan kmp_search, yaitu dengan membuat tabel lps terlebih dahulu. Namun, ketika sebuah kecocokan penuh ditemukan, fungsi ini akan menyimpan indeks posisi tersebut ke dalam sebuah daftar. Setelah itu, alih-alih berhenti, pencarian dilanjutkan dengan menggeser pattern sesuai nilai dari tabel lps untuk menemukan kemungkinan kecocokan berikutnya. Fungsi ini akan mengembalikan sebuah daftar yang berisi semua indeks awal di mana pattern ditemukan.

```
def compute_lps(pattern, m, lps)
```

Fungsi ini adalah langkah preprocessing (pra-pemrosesan) yang krusial untuk algoritma KMP. Tujuannya adalah untuk mengisi sebuah array yang disebut lps (Longest proper Prefix which is also Suffix). Untuk setiap sub-string dari pattern yang dimulai dari awal, fungsi ini menghitung panjang dari awalan (prefix) terpanjang yang juga merupakan akhiran (suffix) dari sub-string tersebut. Informasi inilah yang disimpan dalam array lps dan nantinya digunakan oleh fungsi pencarian KMP untuk melakukan "lompatan" cerdas ketika terjadi ketidakcocokan, sehingga menghindari perbandingan yang tidak perlu.

```
def levenshtein_distance(str1, str2)
```

Fungsi levenshtein_distance menghitung jarak Levenshtein antara dua string, yang merupakan jumlah minimum operasi edit (penyisipan, penghapusan, atau penggantian karakter) yang diperlukan untuk mengubah satu string menjadi string lainnya. Algoritma ini menggunakan pendekatan dynamic programming dengan membuat sebuah matriks untuk menyimpan jarak antara semua prefix dari kedua string. Setiap sel dalam matriks diisi berdasarkan biaya operasi termurah (tambah 1 untuk edit, atau 0 jika karakter sama) dari sel-sel tetangganya. Nilai akhir di sudut kanan bawah matriks adalah jarak total antara kedua string tersebut.

4.2. Pseudocode

Boyer Moore:

```
Procedure BoyerMoore(text, pattern)

Kamus Lokal:
    text       : string
    pattern    : string
    m, n       : integer
    bad_char   : array[0…255] of integer
    s, j       : integer
```

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

    // Kasus tepi
    if m = 0 then
        return 0
    endif
    if m > n then
        return -1
    endif

    // Inisialisasi tabel bad-character
    for i from 0 to 255 do
        bad_char[i] ← -1
    endfor
    for i from 0 to m-1 do
        bad_char[ ord(pattern[i]) ] ← i
    endfor

    // Fase pencarian
    s ← 0
    while s <=n - m do
        j ← m - 1
        // Cocokkan dari kanan ke kiri
        while j >= 0 and pattern[j] = text[s + j] do
            j ← j - 1
        endwhile

        if j < 0 then
            // Ditemukan pada posisi s
            return s
        else
            // Hitung pergeseran menggunakan bad-character rule
            shift ← j - bad_char[ ord(text[s + j]) ]
            if shift < 1 then
                shift ← 1
            endif
            s ← s + shift
        endif
    endwhile

    // Tidak ditemukan
    return -1
EndProcedure


Procedure BoyerMooreAll(text, pattern)

Kamus Lokal:
    text       : string
    pattern    : string
```

```
    m, n       : integer
    bad_char   : array[0…255] of integer
    positions  : list of integer
    s, j       : integer

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

    // Kasus tepi
    if m = 0 then
        return [0]
    endif
    if m > n then
        return []
    endif

    // Inisialisasi tabel bad-character
    for i from 0 to 255 do
        bad_char[i] ← -1
    endfor
    for i from 0 to m-1 do
        bad_char[ ord(pattern[i]) ] ← i
    endfor

    positions ← empty list
    s ← 0

    // Fase pencarian
    while s <= n - m do
        j ← m - 1
        // Cocokkan dari kanan ke kiri
        while j >= 0 and pattern[j] = text[s + j] do
            j ← j - 1
        endwhile

        if j < 0 then
            // Tambahkan posisi kecocokan
            append s to positions

            // Hitung pergeseran berikutnya
            if s + m < n then
                shift ← m - bad_char[ ord(text[s + m]) ]
                if shift < 1 then
                    shift ← 1
                endif
            else
                shift ← 1
            endif
            s ← s + shift
        else
            // Hitung pergeseran pada mismatch
```

```
            shift ← j - bad_char[ ord(text[s + j]) ]
            if shift < 1 then
                shift ← 1
            endif
            s ← s + shift
        endif
    endwhile

    return positions
EndProcedure
```

KMP.py:

```
Procedure KMPSearch(text, pattern)

Kamus Lokal:
    text        : string
    pattern     : string
    n, m        : integer
    lps         : array [0…m-1] of integer
    i, j        : integer

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

    // Kasus tepi
    if m = 0 then
        return 0
    endif
    if m > n then
        return -1
    endif

    // Hitung tabel LPS (longest proper prefix yang juga suffix)
    lps ← array of size m, diisi 0
    Call ComputeLPS(pattern, m, lps)

    // Fase pencocokan
    i ← 0   // indeks text
    j ← 0   // indeks pattern
    while i < n do
        if pattern[j] = text[i] then
            i ← i + 1
            j ← j + 1
        endif

        if j = m then
            // Ditemukan kecocokan lengkap
            return i - j
        elseif i < n and pattern[j] != text[i] then
```

```
            if j != 0 then
                j ← lps[j - 1]
            else
                i ← i + 1
            endif
        endif
    endwhile

    // Tidak ditemukan
    return -1
EndProcedure

Procedure KMPSearchAll(text, pattern)

Kamus Lokal:
    text       : string
    pattern    : string
    n, m       : integer
    lps        : array [0…m-1] of integer
    positions  : list of integer
    i, j       : integer

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

    // Kasus tepi
    if m = 0 then
        return [0]
    endif
    if m > n then
        return []
    endif

    // Hitung tabel LPS
    lps ← array of size m, diisi 0
    Call ComputeLPS(pattern, m, lps)

    positions ← empty list
    i ← 0
    j ← 0

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

        if j = m then
            append (i - j) to positions
            j ← lps[j - 1]
```

```
        elseif i < n and pattern[j] != text[i] then
            if j != 0 then
                j ← lps[j - 1]
            else
                i ← i + 1
            endif
        endif
    endwhile

    return positions
EndProcedure

Procedure ComputeLPS(pattern, m, lps)

Kamus Lokal:
    pattern    : string
    m          : integer
    lps        : array [0…m-1] of integer
    length     : integer
    i          : integer

Algoritma:
    length ← 0         // panjang prefix yang cocok
    lps[0] ← 0
    i ← 1

    while i < m do
        if pattern[i] = pattern[length] then
            length ← length + 1
            lps[i] ← length
            i ← i + 1
        else
            if length != 0 then
                // mundur ke lps sebelumnya tanpa increment i
                length ← lps[length - 1]
            else
                lps[i] ← 0
                i ← i + 1
            endif
        endif
    endwhile
EndProcedure
```

Levenshtein distance:

```
Procedure LevenshteinDistance(str1, str2)

Kamus Lokal:
    str1, str2       : string
    len1, len2       : integer
    dp               : array [0…len1] of array [0…len2] of integer
```

```
    i, j             : integer

Algoritma:
    len1 ← length(str1)
    len2 ← length(str2)

    // Inisialisasi matriks dp
    dp ← 2D array ukuran (len1+1) × (len2+1)
    for i from 0 to len1 do
        for j from 0 to len2 do
            if i = 0 then
                dp[i][j] ← j         // j penyisipan karakter
            elseif j = 0 then
                dp[i][j] ← i         // i penghapusan karakter
            else
                if str1[i-1] = str2[j-1] then
                    dp[i][j] ← dp[i-1][j-1]    // karakter sama, tanpa biaya
                else
                    // Pilih operasi minimal: hapus, sisip, atau ganti
                    dp[i][j] ← 1 + min(
                        dp[i-1][j],    // hapus str1[i-1]
                        dp[i][j-1],    // sisipkan str2[j-1]
                        dp[i-1][j-1]   // ganti str1[i-1] → str2[j-1]
                    )
                endif
            endif
        endfor
    endfor
    return dp[len1][len2]
EndProcedure
```

4.3. Source Code

main.go:

```
import sys
import os

# Add project root to Python path
project_root = os.path.dirname(os.path.abspath(__file__))
sys.path.insert(0, project_root)

def main():
    """Main application entry point"""
    try:
        print("Starting ATS - Applicant Tracking System...")        # Import
and run the GUI application
        from gui.main_gui import main as gui_main
        gui_main()
```

```
    except ImportError as e:
        print(f" Import Error: {e}")
        print("Please ensure all dependencies are installed.")
        print("Run: python scripts/setup.py")
        sys.exit(1)
    except Exception as e:
        print(f" Application Error: {e}")
        sys.exit(1)

if __name__ == "__main__":
    main()
```

levenshtein.py:

```
def levenshtein_distance(str1, str2):
    len_str1 = len(str1)
    len_str2 = len(str2)
    dp = [[0] * (len_str2 + 1) for _ in range(len_str1 + 1)]

    for i in range(len_str1 + 1):
        for j in range(len_str2 + 1):
            if i == 0:
                dp[i][j] = j
            elif j == 0:
                dp[i][j] = i
            elif str1[i - 1] == str2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j -
1])
    return dp[len_str1][len_str2]
```

kmp.py:

```
def kmp_search(text, pattern):
    """
    KMP search algorithm that returns the first occurrence position
    Returns -1 if pattern is not found
    """
    m = len(pattern)
    n = len(text)

    if m == 0:
        return 0
    if m > n:
        return -1

    lps = [0] * m

    # Preprocessing the pattern to create lps array
    j = 0
```

```
    compute_lps(pattern, m, lps)

    # Matching phase
    i = 0
    while i < n:
        if pattern[j] == text[i]:
            i += 1
            j += 1
        if j == m:
            return i - j  # Match found
        elif i < n and pattern[j] != text[i]:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1
    return -1

def kmp_search_all(text, pattern):
    """
    KMP search algorithm that returns all occurrence positions
    Returns list of positions where pattern is found
    """
    m = len(pattern)
    n = len(text)

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

    lps = [0] * m
    positions = []

    # Preprocessing the pattern to create lps array
    j = 0
    compute_lps(pattern, m, lps)

    # Matching phase
    i = 0
    while i < n:
        if pattern[j] == text[i]:
            i += 1
            j += 1
        if j == m:
            positions.append(i - j)  # Match found
            j = lps[j - 1]
        elif i < n and pattern[j] != text[i]:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1
```

```
    return positions

# Helper function to create lps (longest prefix suffix) array
def compute_lps(pattern, m, lps):
    length = 0
    i = 1
    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
```

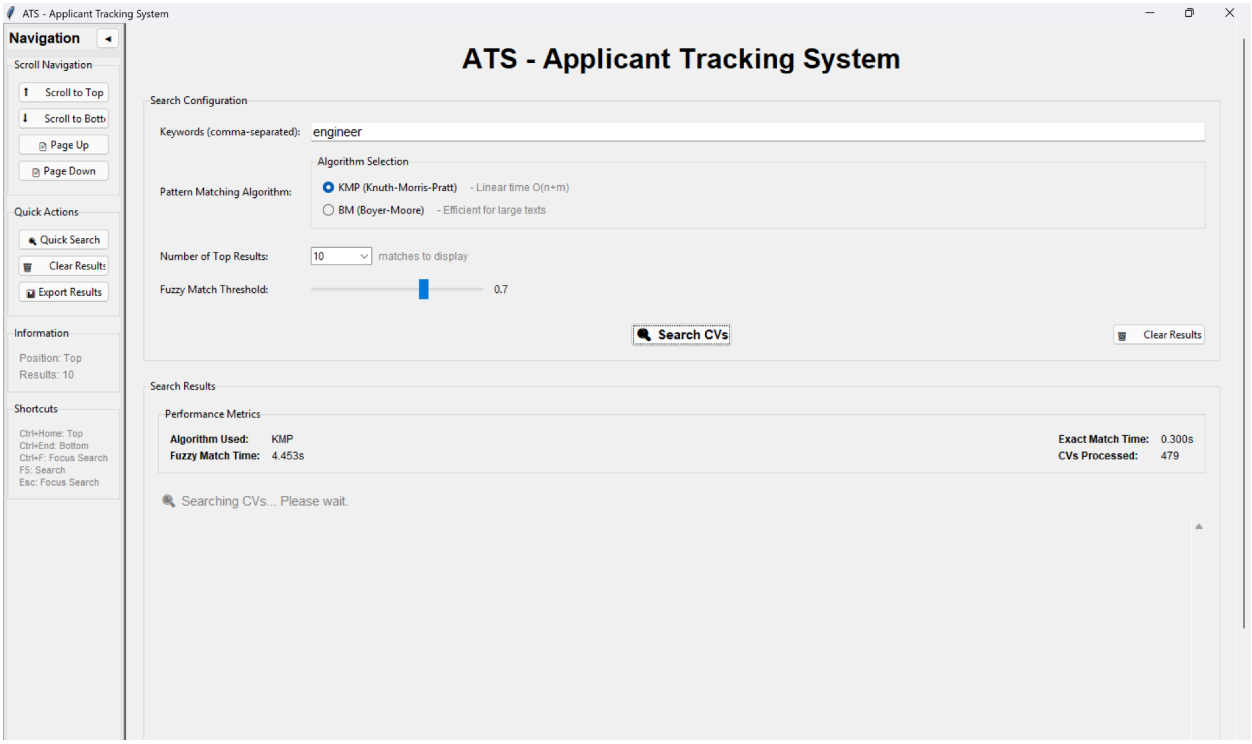
bm.py:

```
def boyer_moore(text, pattern):
    """
    Boyer-Moore search algorithm that returns the first occurrence position
    Returns -1 if pattern is not found
    """
    m = len(pattern)
    n = len(text)

    if m == 0:
        return 0
    if m > n:
        return -1

    bad_char = [-1] * 256

    # Preprocess the pattern
    for i in range(m):
        bad_char[ord(pattern[i])] = i

    # Searching phase
    s = 0
    while s <= n - m:
        j = m - 1
        while j >= 0 and pattern[j] == text[s + j]:
            j -= 1
        if j < 0:
            return s
        else:
            s += max(1, j - bad_char[ord(text[s + j])])
    return -1
```

```
def boyer_moore_all(text, pattern):
    """
    Boyer-Moore search algorithm that returns all occurrence positions
    Returns list of positions where pattern is found
    """
    m = len(pattern)
    n = len(text)

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

    bad_char = [-1] * 256
    positions = []

    # Preprocess the pattern
    for i in range(m):
        bad_char[ord(pattern[i])] = i

    # Searching phase
    s = 0
    while s <= n - m:
        j = m - 1
        while j >= 0 and pattern[j] == text[s + j]:
            j -= 1
        if j < 0:
            positions.append(s)
            s += max(1, m - bad_char[ord(text[s + m])] if s + m < n else 1)
        else:
            s += max(1, j - bad_char[ord(text[s + j])])

    return positions
```

4.4. Penjelasan Tata Cara Penggunaan Program

Berikut merupakan interface yang telah dibangun



Gambar 4.4.1. Interface ATS
(Sumber: Dokumentasi Pribadi)

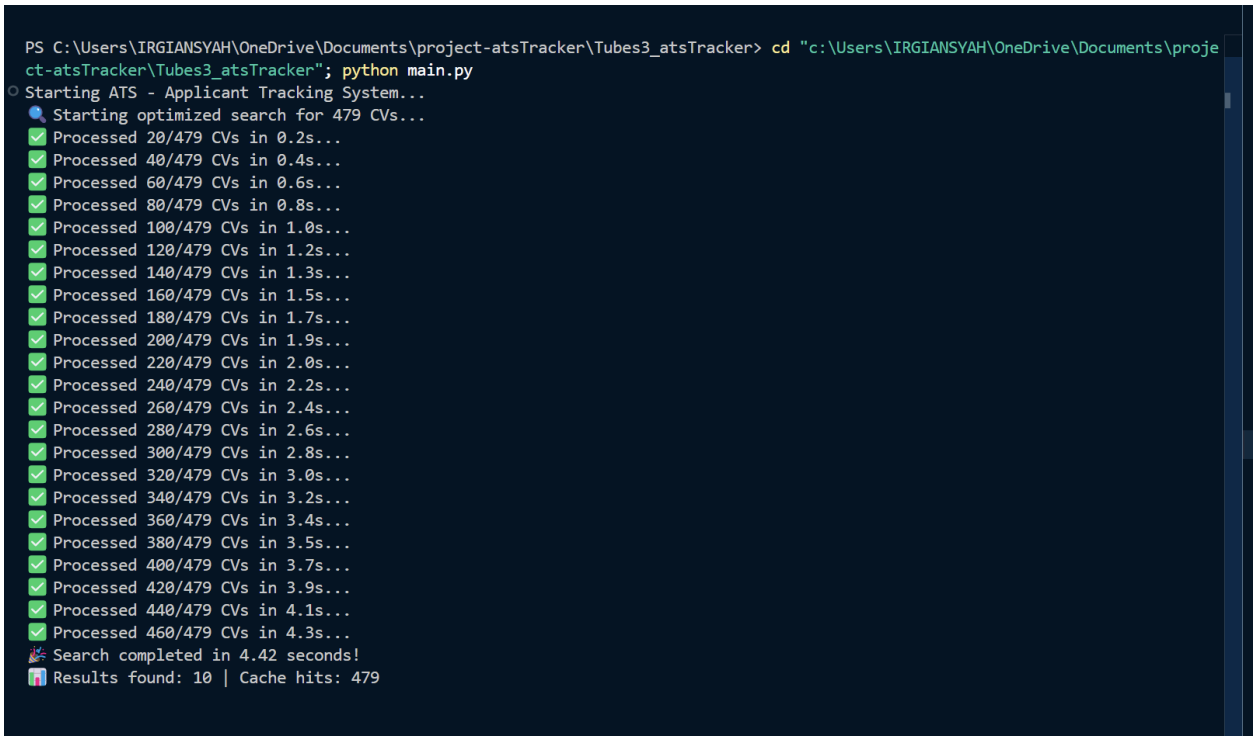
Berikut langkah-langkah penggunaan Program:

Pada interface input pengguna memasuki empat Parameter:

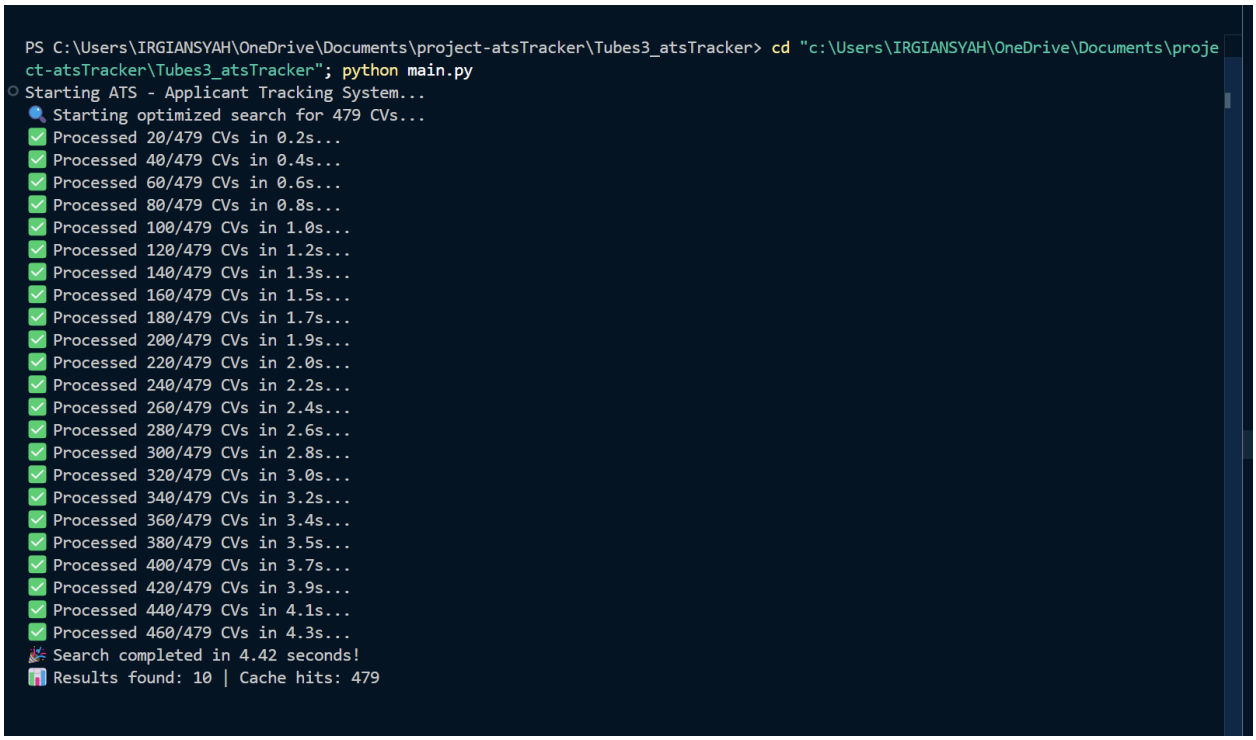
1. Pengguna memasukan keyword yang akan dicari.
2. Pengguna memilih Jumlah algoritma antara “KMP” atau “BM”.
3. Pengguna juga bisa memilih berapa total hasil terbaik yang akan di munculkan.
4. Ketika pengguna sudah mengisi input yang diperlukan, pengguna bisa menekan tombol “Search” dan hasil akan muncul di kolom search result.

4.5. Hasil Pengujian

1. Percobaan 1: KMP, search = “engineer”



2. Percobaan 2: BM, search = “engineer”



4.6. Analisis Hasil Pengujian

Kedua metode matching pattern bisa dengan efektif melacak kata kunci dengan sesuai. Namun, algoritma BM lebih efektif pada string non binary, karena metode pencarian yang lebih sesuai untuk string non-binary. Selanjutnya, informasi tentang CV dapat diperoleh melalui REGEX dan ditampilkan pada user melalui GUI. Lalu, user juga dapat membuka CV secara langsung jika ingin.

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dalam pengembangan sistem Applicant Tracking System (ATS) untuk penyaringan CV, kedua algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) menawarkan keunggulan dalam efisiensi dan kecepatan pencocokan pola, meskipun keduanya memiliki karakteristik yang berbeda. Algoritma KMP dikenal dengan kemampuannya untuk menghindari perbandingan karakter yang berulang, berkat penggunaan tabel Longest Prefix Suffix (LPS) yang memungkinkan pola digeser secara cerdas setelah terjadi ketidakcocokan. Dengan waktu pencarian yang linier (O(n + m)), KMP memberikan performa yang optimal untuk teks berukuran besar meskipun membutuhkan pemrosesan tambahan pada tahap pra-pemrosesan pola. Di sisi lain, Algoritma Boyer-Moore memiliki keunggulan dalam hal kecepatan, terutama pada teks yang panjang. Dengan memanfaatkan teknik bad character rule dan good suffix rule, BM dapat melakukan pergeseran yang lebih jauh saat terjadi ketidakcocokan, sehingga lebih efisien dalam menghindari perbandingan karakter yang tidak perlu. BM seringkali memberikan hasil yang lebih cepat dalam praktik karena dapat melewati sebagian besar teks tanpa perlu membandingkan setiap karakter. Meskipun demikian, algoritma ini memerlukan dua tahap pra-pemrosesan yang lebih kompleks dibandingkan KMP.

Secara keseluruhan, KMP lebih cocok digunakan pada sistem ATS yang menangani teks dalam ukuran lebih kecil atau ketika implementasi yang lebih sederhana dibutuhkan, sementara Boyer-Moore lebih efektif untuk pencocokan pola dalam teks yang sangat panjang, seperti CV dengan banyak data, berkat teknik pergeseran yang efisien. Pemilihan algoritma yang tepat antara KMP dan BM sangat bergantung pada kebutuhan aplikasi, khususnya terkait ukuran teks yang akan diproses dan kompleksitas implementasi yang diinginkan. Keduanya merupakan pilihan yang solid untuk meningkatkan kecepatan dan akurasi pencocokan dalam sistem ATS.

5.2. Saran

Tugas besar sudah baik, hanya saja waktunya cukup padat dan penuh dengan tubes lain dan ujian lain.

5.3. Refleksi

Pengerjaan Tugas Besar Strategi Algoritma sebaiknya dilakukan jauh-jauh hari sebelum deadline. Sehingga, pada saat waktu pengumpulan, pekerjaan tetap bisa berjalan lancar tanpa mengalami masalah. Dan bisa mendapat hasil yang lebih maksimal.

LAMPIRAN

Github	https://github.com/irgimondo/Tubes3_atsTracker

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

DAFTAR PUSTAKA

Munir, R. (2024). *Pencocokan string (string matching) dengan algoritma brute force, KMP, Boyer-Moore*. Diakses pada 9 Mei 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf

Munir, R. (2024). *Pencocokan string dengan regular expression (regex)*. Diakses pada 10 Mei 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf