IF2211 - STRATEGI ALGORITMA

LAPORAN TUGAS BESAR 3

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

Disusun Oleh:

Kelompok 23 - SigningOut

Najwa Kahani Fatima	13523043
Nayla Zahira	13523079
Heleni Gratia M Tampubolon	13523107

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

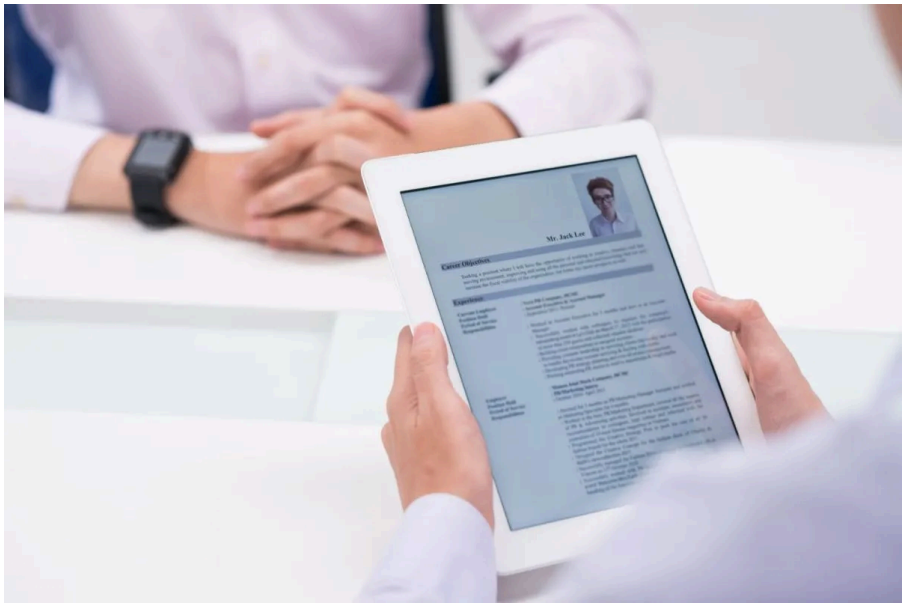
JL. GANESA 10, BANDUNG, 40132

2025

DAFTAR ISI

BAB I

DESKRIPSI MASALAH



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

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

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

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

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

BAB II

LANDASAN TEORI

2.1. Exact String Matching

Exact string matching merupakan proses pencarian pola (*pattern*) dalam sebuah teks (*text*) untuk menemukan semua kemunculan pola tersebut secara tepat dan berurutan. Hal ini berarti setiap karakter dalam pola harus sesuai secara identik dengan urutan karakter dalam bagian tertentu dari teks. Masalah ini merupakan salah satu permasalahan dasar dalam ilmu komputer dan memiliki banyak aplikasi, seperti pencarian kata kunci dalam dokumen, pemeriksaan DNA dalam bioinformatika, hingga fitur pencarian pada editor teks dan web browser.

Secara formal, diberikan sebuah teks T dengan panjang n dan pola P dengan panjang m, tujuan dari exact string matching adalah menemukan semua posisi i dalam T sehingga substring T[i..i+m-1] sama persis dengan P[0..m-1].

Metode paling sederhana untuk menyelesaikan masalah ini adalah *brute-force*, yaitu mencocokkan pola dengan setiap posisi dalam teks satu per satu. Meskipun mudah diimplementasikan, pendekatan ini memiliki kompleksitas waktu O(n×m), yang kurang efisien untuk data berskala besar.

Untuk meningkatkan efisiensi, berbagai algoritma pencocokan pola yang lebih canggih telah dikembangkan, di antaranya:

1. Knuth-Morris-Pratt (KMP)

   Algoritma yang menghindari perbandingan ulang dengan memanfaatkan informasi dari pola itu sendiri melalui prefix-suffix table (LPS), sehingga cocok untuk pencarian tunggal dengan performa O(n + m).

2. Boyer-Moore

   Algoritma untuk melakukan pencocokan dari kanan ke kiri dan menggunakan dua heuristik utama (bad character dan good suffix) untuk menggeser pola sejauh mungkin. Algoritma ini sangat cepat terutama untuk alfabet besar dan teks panjang.

3. Aho-Corasick

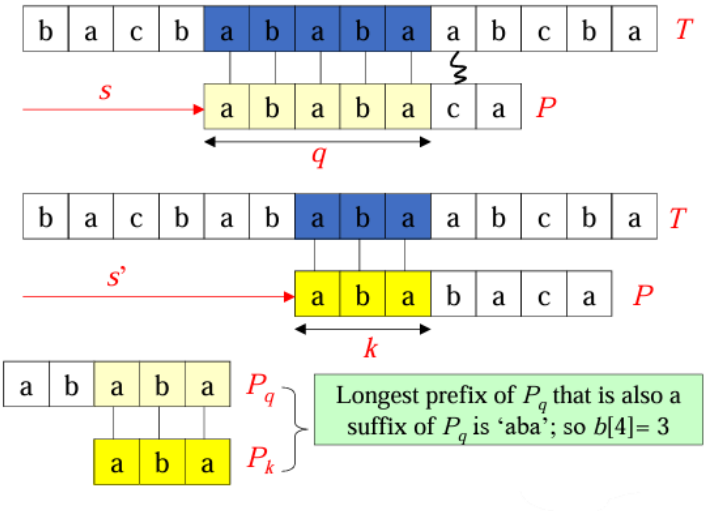
   Algoritma ini digunakan untuk mencari banyak pola sekaligus dalam satu teks. Pertama, semua pola disusun dalam sebuah struktur mirip pohon (*trie*). Setelah itu, algoritma membangun jalur alternatif (*fail links*) agar saat pencocokan gagal, pencarian bisa

langsung melompat ke bagian yang mungkin cocok berikutnya tanpa harus mengulang dari awal. Proses pencarian dilakukan hanya sekali membaca teks, sehingga sangat efisien. Aho-Corasick cocok digunakan dalam program seperti pencarian kata kunci di dokumen atau sistem deteksi kata-kata sensitif di teks secara cepat dan serentak.

2.2. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) merupakan algoritma pencarian string yang efisien untuk menemukan kemunculan suatu pola (pattern) di dalam sebuah teks. Algoritma ini bekerja dengan melakukan praproses terhadap pola untuk membentuk sebuah tabel yang dikenal sebagai *Longest Prefix Suffix* (LPS) atau fungsi pinggiran (*border function*). Fungsi ini, yang dinyatakan sebagai b(k), merepresentasikan panjang prefiks terbesar dari P[0..k] yang juga merupakan sufiks dari P[1..k]. Praproses ini memungkinkan algoritma untuk menghindari pemeriksaan ulang karakter pada teks yang telah dicocokkan sebelumnya saat terjadi ketidakcocokan.

Dalam proses pencarian, KMP memindai teks dari kiri ke kanan seperti algoritma brute force, tetapi dengan strategi pergeseran pola yang jauh lebih efisien. Ketika terjadi ketidakcocokan pada posisi P[j] (yakni saat T[i] ≠ P[j]), pola akan digeser berdasarkan nilai pada tabel LPS, bukan secara satu per satu seperti pada pencocokan naive. Hal ini membuat proses pencarian menjadi lebih cepat dan tidak memerlukan pergerakan mundur dalam teks, yang sangat bermanfaat saat memproses data berukuran besar seperti file teks atau aliran data.



Gambar 2. Ilustrasi Pencarian String dengan Algoritma KMP
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

Algoritma ini memiliki kompleksitas waktu sebesar O(m + n), dengan m adalah panjang pola dan n adalah panjang teks, di mana O(m) digunakan untuk menghitung fungsi LPS, dan O(n) untuk pencarian pola dalam teks.

Meskipun efisien dan unggul dalam menghindari pencocokan ulang karakter, algoritma KMP dapat menunjukkan performa yang sedikit menurun ketika pola memiliki tingkat keragaman alfabet yang tinggi, terutama jika ketidakcocokan sering terjadi di bagian akhir pola. Dalam kasus tersebut, nilai-nilai LPS cenderung kecil sehingga pergeseran pola menjadi minimal dan mendekati perilaku brute force.

Algoritma tersebut dapat dituliskan dalam Pseudocode sebagai berikut.

```
procedure KMPMatch(input text: string, input pattern: string)
{
Menemukan kemunculan pertama pattern dalam text menggunakan algoritma KMP
Masukan: text dan pattern berupa string
Keluaran: indeks awal kecocokan atau -1 jika tidak ditemukan
}

Deklarasi:
n, m : integer
i, j : integer
b[0..m-1] : array of integer

Algoritma:
n ← panjang(text)
m ← panjang(pattern)
b ← ComputeBorder(pattern)
i ← 0
j ← 0

while i < n do
    if pattern[j] = text[i] then
        if j = m - 1 then
            return i - m + 1 // Kecocokan ditemukan
        endif
        i ← i + 1
        j ← j + 1
    else if j > 0 then
        j ← b[j - 1]
    else
        i ← i + 1
    endif
endwhile

→ -1 // Tidak ditemukan

procedure ComputeBorder(input pattern: string) → array of integer
{
Membangun fungsi prefix (Longest Prefix Suffix) dari pattern
```

```
Masukan: pattern berupa string
Keluaran: array border function
}

Deklarasi:
m : integer
i, j : integer
b[0..m-1] : array of integer

Algoritma:
m ← panjang(pattern)
b[0] ← 0
i ← 1
j ← 0

while i < m do
    if pattern[i] = pattern[j] then
        b[i] ← j + 1
        i ← i + 1
        j ← j + 1
    else if j > 0 then
        j ← b[j - 1]
    else
        b[i] ← 0
        i ← i + 1
    endif
endwhile

→ b
```

2.3. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah algoritma pencocokan pola yang sangat efisien dan digunakan secara luas dalam berbagai aplikasi pencarian teks. Algoritma ini bekerja dengan mencocokkan pola P dalam teks T dari kiri ke kanan, tetapi melakukan pencocokan karakter dari belakang pola (kanan ke kiri). Keunggulan utamanya terletak pada kemampuannya untuk melompati bagian teks yang tidak mungkin cocok, sehingga mengurangi jumlah perbandingan secara drastis.

Pendekatan Boyer-Moore didasarkan pada dua teknik utama:

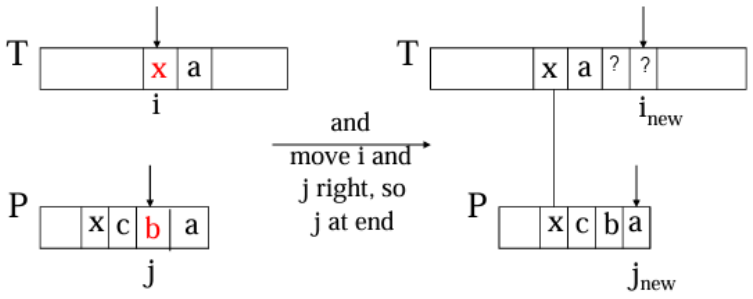
1. Looking-Glass Technique (Teknik Cermin)

   Pencocokan karakter dilakukan dari akhir pola ke awal. Ketika terjadi ketidakcocokan, algoritma segera menghentikan pencocokan dan memutuskan seberapa jauh pola dapat digeser berdasarkan informasi yang telah diproses sebelumnya.

2. Character-Jump Technique (Teknik Lompatan Karakter)

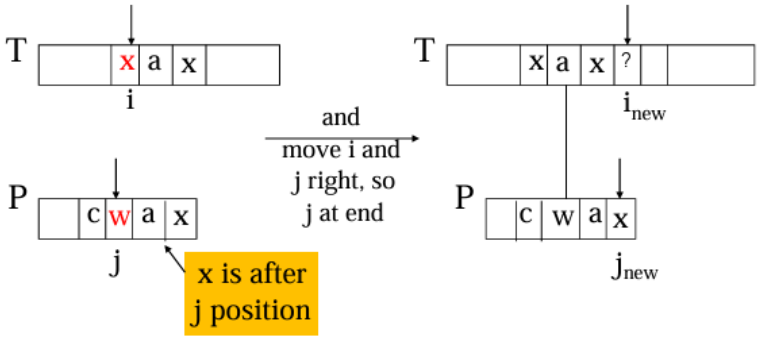
Ketika terjadi ketidakcocokan pada karakter T[i] = x dan P[j] ≠ x, algoritma menggunakan informasi kemunculan karakter untuk menentukan arah dan jarak geseran pola. Tiga kemungkinan kasus berikut digunakan:

- Case 1: Jika karakter x terdapat dalam pola P, geser P ke kanan untuk menyejajarkan kemunculan terakhir x dalam P dengan posisi T[i].



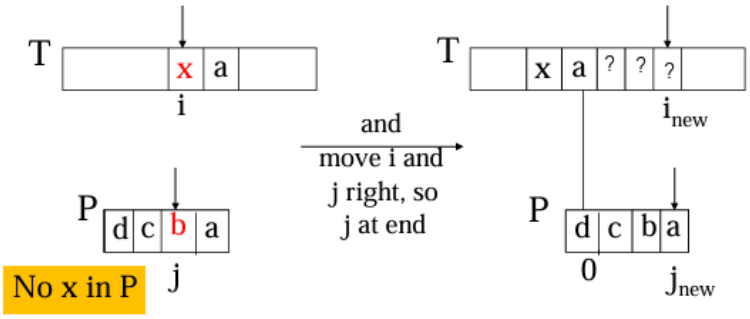
Gambar 3. Ilustrasi Kasus 1 Pencarian String dengan Algoritma BM
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

- Case 2: Jika karakter x ada dalam P namun tidak memungkinkan untuk menyelaraskan karena posisinya lebih kanan dari j, geser P ke kanan sejauh 1 karakter, sehingga sejajar dengan T[i+1].



Gambar 4. Ilustrasi Kasus 2 Pencarian String dengan Algoritma BM
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

- Case 3: Jika x tidak ada dalam pola P, geser P sehingga P[0] sejajar dengan T[i+1].



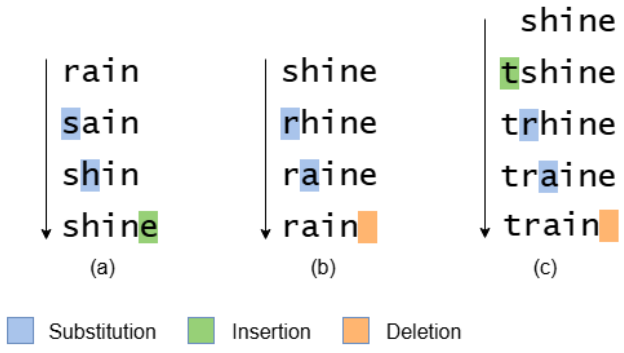
Gambar 5. Ilustrasi Kasus 3 Pencarian String dengan Algoritma BM
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

Untuk mendukung strategi ini, algoritma Boyer-Moore menggunakan struktur data yang disebut last occurrence function, dilambangkan dengan L(x), yang menyimpan indeks terakhir kemunculan setiap karakter x dalam pola P, atau -1 jika karakter tersebut tidak muncul.

Selain *bad character heuristic*, algoritma ini juga dapat dilengkapi dengan *good suffix heuristic*, yaitu strategi tambahan yang mempertimbangkan bagian pola yang telah cocok untuk menentukan pergeseran lebih jauh. Karena strategi geseran yang cerdas ini, Boyer-Moore sangat efisien, terutama untuk teks berukuran besar dan alfabet yang besar, menjadikannya salah satu algoritma pencarian pola tercepat yang tersedia.

2.4. Fuzzy Matching: Levenshtein Distance

Levenshtein Distance adalah metode yang digunakan dalam ilmu komputer untuk mengganti kata yang salah eja dengan kata kamus yang benar dengan menghitung jumlah minimum penyuntingan karakter tunggal yang diperlukan untuk mengubah satu kata menjadi kata lain. Dengan metode ini, dapat diukur tingkat persamaan antara dua string yang melibatkan penambahan karakter, penghapusan karakter, atau substitusi karakter.



Gambar 6. Ilustrasi Levenshtein Distance
Sumber: https://devopedia.org/levenshtein-distance\

Pengukuran jarak levenshtein pada program ini menggunakan pendekatan *dynamic programming* dengan menyimpan biaya yang menyatakan banyak perubahan pada pengecekan karakter sejauh n dan mengambil nilai perubahan terkecil jika karakter tidak sama.

Mula-mula dibuat dua array kosong (setelah ini disebut array biaya) dengan panjang m, yakni panjang string terbesar untuk menyimpan biaya perubahan. Lalu, dilakukan perbandingan untuk setiap karakter pada string yang lebih pendek dengan karakter-karakter pada string yang lebih panjang. Jika karakter sama, maka tidak ada perubahan pada array biaya. Jika karakter tidak sama pada iterasi ke-k pada string yang lebih panjang, akan ditambah biaya sebesar 1 dari biaya minimum yang dibutuhkan untuk mengubah string 1 menjadi string 2, lalu menyimpannya pada array biaya saat ini *(curr)*. Pada setiap iterasi, array biaya sebelumnya *(prev)* akan diperbarui dari array biaya saat ini *(curr)*.

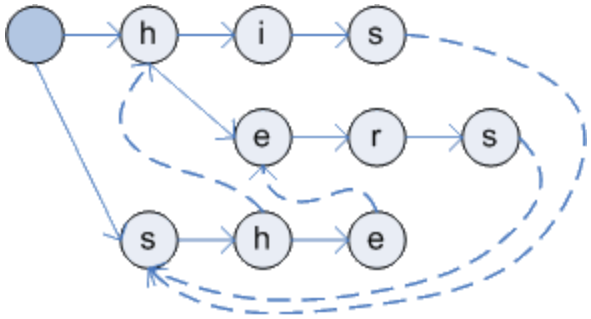
```
procedure LevenshteinDistance(input string1: string, input string2: string)
{
Menghitung jarak Levenshtein antara dua string
Masukan : string1 dan string2
Keluaran: integer yang menyatakan jarak Levenshtein
}

Deklarasi:
m, n : integer
i, j : integer
longer, shorter : string
prev[0..m] : array of integer
curr[0..m] : array of integer

Algoritma:
if panjang(string1) > panjang(string2) then
    longer ← string1
    shorter ← string2
else
    longer ← string2
    shorter ← string1
endif

m ← panjang(longer)
n ← panjang(shorter)

if m = 0 then
    return n
endif
if n = 0 then
    return m
endif

// Inisialisasi array prev
for i ← 0 to m do
    prev[i] ← i
```

```
endfor

// Inisialisasi array curr
for i ← 0 to m do
    curr[i] ← 0
endfor

for j ← 1 to n do
    curr[0] ← j

    for k ← 1 to m do
        if shorter[j - 1] = longer[k - 1] then
            curr[k] ← prev[k - 1] // karakter sama, tidak ada biaya
        else
            curr[k] ← 1 + minimum dari:
                - curr[k - 1]  // insert
                - prev[k]      // remove
                - prev[k - 1]  // replace
        endif
    endfor

    // Copy curr ke prev
    for i ← 0 to m do
        prev[i] ← curr[i]
    endfor
endfor

→ curr[m]
```

2.5. Algoritma Aho-Corasick

Aho-corasick merupakan algoritma string matching dengan memanfaatkan pohon dalam pencatatan *keywoard* (struktur data trie) sehingga mempermudah pencarian tanpa perlu melakukan iterasi perbandingan pencocokan untuk setiap *keywoard* yang ada. Pengecekan dilakukan dengan menggunakan *failure function* yang dibangun di awal pencarian.

Pencarian kata dilakukan dengan melakukan pengecekan pada karakter pertama pada root trie. Jika cocok, maka pengecekan dilanjutkan ke karakter selanjutnya pada jalur tersebut. Jika tidak, maka program akan mencari karakter berbeda selanjutnya yang telah dikonfigurasi pada *failure function* (pada ilustrasi di bawah ditunjukkan dengan garis putus-putus). Oleh karena itu, jika suatu proses pencocokan gagal di tengah kata, pencarian tidak akan dilakukan ulang, namun diteruskan berdasarkan *failure function.*



Gambar 7. Ilustrasi Pencarian Menggunakan Keyword Tree
Sumber: https://tomasp.net/blog/ahocorasick.aspx/

Algoritma pembuatan *failure function*:

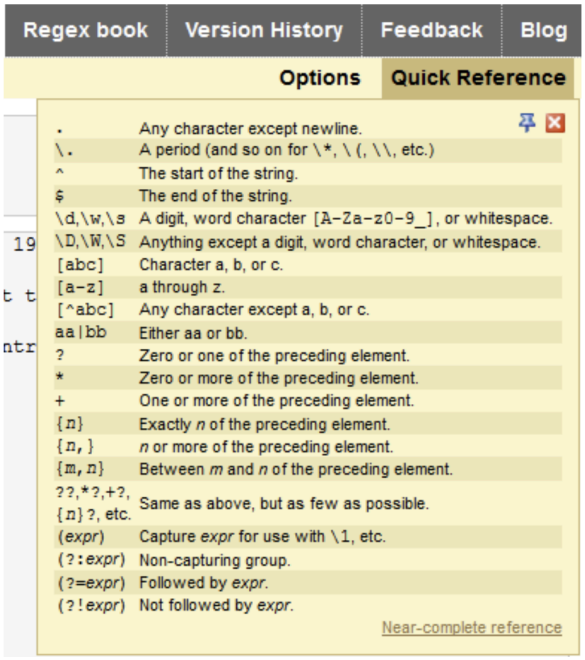
```
Procedure BuildFailureFunction(trie)
Input: trie - akar dari trie berisi semua pattern
Output: failure - map dari node ke node fallback (failure link)

Deklarasi:
    queue ← kosong (FIFO queue)
    failure ← map kosong

Algoritma:
// Inisialisasi failure link untuk semua child langsung dari root:
    for each child_node of root:
        failure[child_node] ← root
        enqueue(queue, child_node)

// Proses node lain secara BFS:
    while queue tidak kosong:
        current_node ← Dequeue(queue)

        for each char → child_node in current_node:
            if char == '$' (penanda akhir kata):
                continue

            enqueue(queue, child_node)

            # mencari fallback node dari current_node
            fallback ← failure[current_node]

            while fallback ≠ root and char not in fallback:
                fallback ← failure[fallback]

            if char in fallback:
                failure[child_node] ← fallback[char]
            else:
                failure[child_node] ← root

→ failure
```

Algoritma pencarian Aho-Corasick:

```
Procedure AhoCorasickSearch(input text: string)
{
  Mencari semua kemunculan kata kunci dalam text menggunakan automaton
Aho-Corasick
  Masukan : text (string)
  Keluaran: result_counts (dictionary jumlah kemunculan setiap kata kunci)
}

Deklarasi:
  result_counts : dictionary dari string ke integer
  current_node : node (pointer pada trie)
  i : integer
  char : karakter

Algoritma:

Jika struktur failure function belum dibangun
     panggil build_failure_function()

Inisialisasi result_counts dengan setiap kata kunci bernilai 0

Set current_node ← root trie

Untuk setiap karakter (char) dalam text (dengan indeks i):
    Selama current_node bukan root dan char tidak ada sebagai child:
        - current_node ← node hasil dari failure link pada current_node

    Jika char ada dalam current_node:
        - current_node ← current_node[char] (gerak ke child)
        - Dapatkan semua kata kunci yang berakhir di current_node
        - Untuk setiap match dalam output node tersebut:
            - Tambahkan 1 pada result_counts[match]

6. Kembalikan result_counts
```

2.6. Regular Expression (Regex)

Regular Expression adalah pola yang digunakan untuk mencocokkan kombinasi karakter dalam string. Pola pada regular expression dibuat dengan notasi umum regex seperti gambar di bawah ini. Pencarian dilakukan dengan melakukan *matching* pada pola regex yang telah dibuat dan dengan kalimat-kalimat pada persoalan.



Gambar 8. Notasi Umum Regex
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf

2.7. Pengembangan Aplikasi

Program SignHire merupakan aplikasi desktop berbasis python yang menggunakan framework CustomTkinter untuk membangun antarmuka grafis pengguna. Arsitektur yang digunakan adalah Model-View-Controller (MVC), yang memisahkan logika pemrosesan data (model), tampilan antarmuka pengguna (*view*), dan pengendali alur komunikasi serta interaksi pengguna (*controller*).

2.7.1 General User Interface

Antarmuka Pengguna Umum *(General User Interface*) merupakan bagian dari sistem perangkat lunak yang memungkinkan pengguna berinteraksi langsung dengan aplikasi melalui elemen-elemen visual. Dalam program ini, antarmuka pengguna dirancang sebagai jembatan antara pengguna dan proses pencocokan data yang berlangsung di latar belakang.

Antarmuka program mencakup beberapa komponen utama, antara lain:

- Judul aplikasi,
- Tombol toggle untuk memilih algoritma pencocokan (KMP atau Boyer-Moore),
- Selector jumlah hasil (top matches),
- Tombol pencarian (Search),

- Ringkasan hasil pencarian (*summary result*), dan
- Container yang menampilkan daftar data hasil pencocokan berdasarkan kata kunci yang dimasukkan pengguna.

Setiap hasil pencarian ditampilkan dalam bentuk kartu yang berisi informasi sebagai berikut:

- Nama kandidat,
- Jumlah kecocokan (dihitung berdasarkan jumlah kata kunci yang ditemukan),
- Daftar kata kunci yang cocok beserta frekuensi kemunculannya.

Selain itu, pengguna juga dapat melihat ringkasan informasi (summary) yang telah diekstrak dari CV, atau menekan tombol "View CV" untuk membuka file CV asli.

Penjelasan mengenai alur penggunaan program, ialah:

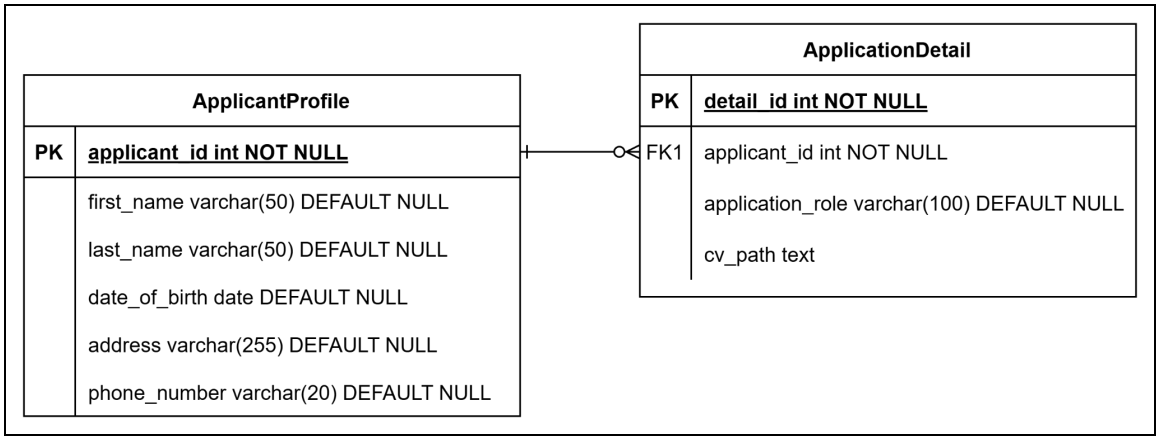
1. Pengguna memasukkan kata kunci pencarian.
2. Memilih algoritma pencocokan: KMP atau Boyer-Moore (BM).
3. Menentukan jumlah hasil yang ingin ditampilkan.
4. Menekan tombol Search.
5. Sistem akan menampilkan daftar CV yang paling relevan, dilengkapi tombol untuk melihat ringkasan informasi atau CV asli.

Sebagai tambahan, program ini juga dilengkapi dengan halaman penjelasan program (About the App) serta profil pengembang (Developer), yaitu tim SigningOut, yang bertanggung jawab atas pengembangan aplikasi ini.

2.7.2 Database

Program ini terhubung dengan database MySQL yang menyimpan dataset CV dalam format terstruktur. Database ini terdiri atas dua tabel utama, yaitu ApplicantProfile dan ApplicationDetail. Tabel ApplicantProfile menyimpan informasi pribadi dari setiap pelamar, seperti nama, tanggal lahir, kontak, dan alamat. Sementara itu, tabel ApplicationDetail mencatat detail setiap cv yang diajukan oleh pelamar, termasuk posisi yang dilamar dan path dari cv.

Relasi antara kedua tabel tersebut adalah one-to-many, di mana satu pelamar yang tercatat dalam tabel ApplicantProfile dapat memiliki beberapa entri lamaran dalam tabel ApplicationDetail. Hal ini karena seorang pelamar dapat mengajukan lamaran ke berbagai posisi. Berikut ini adalah ilustrasi skema relasi basis datanya:



Gambar 9. Notasi Umum Regex
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf

2.8. Kakas dan Bahasa Pemrograman

Berbagai bahasa pemrograman dan kakas yang ada dapat digunakan secara bersamaan untuk menciptakan program yang sesuai dengan kebutuhan. Pada bagian ini, akan dijelaskan beberapa kakas dan bahasa pemrograman yang digunakan.

2.8.1 Python

Python adalah bahasa pemrograman tingkat tinggi yang bersifat interpreted, dinamis, dan memiliki sintaks yang sederhana sehingga mudah dipelajari dan digunakan. Python mendukung berbagai paradigma pemrograman, termasuk pemrograman prosedural, berorientasi objek, dan fungsional. Keunggulan Python salah satunya ialah mendukung ekosistem pustaka (*library*) yang beragam.

Dalam program yang dibangun ini, Python dipilih karena kemampuannya sangat baik dalam mengelola dan memproses data berbasis teks, terutama untuk implementasi algoritma pencocokan string seperti Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Python juga menyediakan modul bawaan seperti re untuk regular expression yang mempermudah proses pencarian kata kunci, serta tkinter yang memungkinkan pembuatan antarmuka grafis secara cepat dan efisien tanpa perlu menggunakan framework eksternal. Dukungan terhadap berbagai format file, seperti PDF juga menjadi pertimbangan penting karena program ini membutuhkan proses ekstraksi informasi dari file CV.

2.8.2 Custom Tkinter

CustomTkinter adalah pustaka Python berbasis tkinter yang menyediakan tampilan antarmuka grafis (GUI) modern dengan desain yang lebih estetis dan fleksibel dibandingkan tkinter standar. Dibangun di atas kerangka kerja tkinter asli, CustomTkinter

memungkinkan pengembang untuk membuat antarmuka pengguna dengan gaya modern UI, seperti tombol berpenampilan datar, warna gelap/terang (dark/light mode), elemen-elemen yang dapat dikustomisasi, dan tampilan yang lebih konsisten di berbagai platform. Pustaka ini sangat cocok digunakan untuk aplikasi desktop yang mengutamakan kemudahan penggunaan serta tampilan yang menarik secara visual, tanpa harus menggunakan framework GUI eksternal yang lebih kompleks seperti PyQt atau Kivy.

Dalam program ini, CustomTkinter digunakan untuk membangun antarmuka pengguna yang interaktif dan nyaman digunakan. Dengan komponen-komponen seperti tombol, kotak input, dan frame kontainer yang dapat disesuaikan tampilannya, CustomTkinter memberikan pengalaman pengguna yang lebih profesional dan intuitif. Penggunaan pustaka ini juga mempermudah pengembang dalam mengatur tema, ukuran elemen, serta responsivitas antarmuka, yang sangat penting dalam menyajikan hasil pencarian secara rapi dan informatif. Dengan demikian, CustomTkinter berperan penting dalam menjembatani fungsionalitas algoritma pencocokan dengan tampilan yang ramah pengguna.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

Permasalahan yang akan diselesaikan pada Tugas Besar ini adalah pencarian kata kunci dan pembuatan ringkasan pada Curriculum Vitae berformat .pdf dengan menggunakan Regular Expression. Untuk pencarian kata kunci, dilakukan dua model *string matching,* yakni exact string matching dengan algoritma 1) Knuth-Morris-Pratt (KMP), 2) Boyer-Moore (BM), dan 3) Aho-Corasick, serta *fuzzy matching* dengan Levenshtein Distance. Pencarian dengan *fuzzy matching* dilakukan ketika pencarian dengan *exact matching* tidak memberikan hasil sejumlah input yang diminta atau ketika tidak ditemukan *exact match* sama sekali.

Pengguna akan memasukkan kata kunci yang ingin dicari pada box pencarian. Kata kunci dapat dipisahkan dengan tanda koma. Setelah itu, pengguna dapat memilih algoritma yang ingin digunakan dalam pencarian (KMP, BM, dan Aho-Corasick) dan jumlah CV yang ingin hasil pencarian yang ingin ditampilkan. Ketika tombol cari di-klik, program akan melakukan *data retrieval* dari database. Lalu, dokumen CV akan diproses menjadi *long string* dengan kosa kata dari input pengguna yang telah di-*parse* menjadi *list of keywords* untuk kemudian diproses menggunakan algoritma yang dipilih.

Setelah itu, program akan memunculkan kartu berisi hasil pencarian berisi nama pelamar, *keyword* yang didapat, dan jumlahnya. Pengguna juga dapat melihat dokumen CV (.pdf) dan ringkasan *(summary)* masing-masing pelamar yang muncul.

Ketika pengguna menekan tombol *summary,* program akan melakukan konversi dokumen CV ke string berformat untuk dilakukan proses *regex* matching. Pada proses ini, program akan mengekstrak informasi *skill*, riwayat pendidikan, dan pengalaman kerja dari pelamar. Hasil dari *regex matching* ini ditampilkan bersamaan dengan informasi pribadi lain yang didapat dari *query* pada database.

Program ini diimplementasikan dalam bentuk aplikasi desktop dengan bahasa Python.

3.2 Proses Pemetaan Masalah

Proses pemetaan masalah untuk menemukan banyaknya kemunculan keyword berbeda-beda tergantung pada algoritma pencocokan yang digunakan. Secara umum, proses pencarian dilakukan dengan parameter input *long string* yang telah diekstrak dari dokumen CV. Berikut adalah proses pemetaan secara umum untuk *string matching* dan *regular expresssion*:

3.2.1 Pemetaan Masalah Knuth-Morris-Pratt

Sebelum melakukan pencarian, algoritma KMP membangun border function array (*failure function*) yang menyimpan informasi tentang prefix dan suffix terpanjang yang sama untuk setiap posisi dalam pattern. Border function ini dihitung dengan melakukan iterasi pada pattern, membandingkan setiap karakter dengan karakter pada posisi yang ditunjuk oleh length pointer, dan menyimpan panjang border untuk setiap posisi dalam array bf.

Setelah border function terbentuk, dilakukan proses pencarian dengan menggunakan dua pointer yaitu i untuk indeks teks dan j untuk indeks pattern. Algoritma membandingkan karakter pada posisi i dan j, jika sama maka kedua pointer dimajukan. Ketika j mencapai panjang pattern, berarti ditemukan kecocokan dan posisi match disimpan, kemudian j direset menggunakan nilai dari *border function* untuk melanjutkan pencarian tanpa mengulang perbandingan yang tidak perlu. Jika karakter tidak cocok, algoritma menggunakan informasi dari border function untuk menggeser pattern secara efisien tanpa mundur pada pointer i, sehingga mencapai kompleksitas waktu O(n + m). Setelah pencarian selesai, akan diperoleh array berisi semua posisi kemunculan kata kunci dalam teks CV, yang kemudian dihitung jumlahnya untuk menentukan relevansi CV terhadap kata kunci tersebut.

3.2.2 Pemetaan Masalah Boyer-Moore

Pencarian kata kunci menggunakan algoritma Boyer-Moore dimulai dengan membangun tabel *last occurrence* yang menyimpan posisi terakhir setiap karakter dalam pattern. Algoritma ini bekerja dengan membandingkan pattern dari kanan ke kiri (suffix matching) pada teks input, dimulai dari posisi yang sejajar dengan akhir pattern. Parameter input pertama adalah teks yang akan dicari dan parameter input kedua adalah pattern kata kunci yang dicari.

Proses pencarian dilakukan dengan menggeser pattern sepanjang teks menggunakan dua fase pengecekan. Ketika terjadi ketidakcocokan karakter, algoritma akan menghitung nilai shift berdasarkan bad character rule yang memanfaatkan tabel last occurrence untuk menentukan seberapa jauh pattern dapat digeser tanpa melewatkan kemungkinan match. Jika karakter yang tidak cocok tidak ada dalam pattern, maka pattern akan digeser melewati posisi karakter tersebut. Jika karakter tersebut ada dalam pattern, maka pattern akan digeser sehingga karakter terakhir yang sama dalam pattern sejajar dengan karakter yang tidak cocok di teks. Proses ini diulang hingga seluruh teks telah diperiksa, dan setiap posisi yang ditemukan akan disimpan sebagai hasil pencarian.

3.2.3 Pemetaan Masalah Aho-Corasick

Algoritma Aho-Corasick adalah algoritma pencarian string yang efisien untuk menemukan banyak pola (kata kunci) sekaligus dalam sebuah teks. Pencarian kemunculan kata kunci dengan Aho-Corasick dimulai dengan pembuatan kelas AhoCorasick(). Inisiasi kelas dilakukan dengan pembuatan variabel-variabel kosong, yaitu trie, map *failure*, output, dan list kata kunci. Cara kerja algoritma ini dimulai dengan membangun struktur trie (pohon prefix) dari seluruh kata kunci yang ingin dicari. Setelah itu, trie dilengkapi dengan *failure function* mirip seperti dalam algoritma Knuth-Morris-Pratt (KMP), yang memungkinkan pencarian berpindah ke node *fallback* saat pencocokan gagal, tanpa mengulang dari awal. Proses ini membuat pencarian tetap efisien meskipun terdapat banyak pola.

Selama proses pencarian di dalam teks, algoritma menjelajahi karakter demi karakter dan mengikuti transisi dalam trie. Bila karakter tidak ditemukan di node saat ini, algoritma mengikuti *failure link* hingga menemukan transisi yang sesuai atau kembali ke akar trie. Setiap kali mencapai node dengan tanda akhir kata ($), atau memiliki output tambahan dari *failure links*, algoritma mencatat kemunculan pola di posisi tersebut. Dengan pendekatan ini, Aho-Corasick mampu melakukan pencarian semua pola sekaligus dalam waktu linear terhadap panjang teks + total panjang kata kunci, yang sangat cocok untuk pencarian dalam CV.

3.2.4 Pemetaan Masalah Levenshtein Distance

Untuk setiap kata kunci, akan dilakukan penghitungan levenshtein distance dengan masing-masing kata pada long string. Parameter input pertama adalah string dari long string dan parameter input kedua adalah string dari kata kunci. Kalkulasi levenshtein distance dilakukan dengan mencari ukuran masing-masing string. Lalu dibuat dua array untuk menyimpan kalkulasi biaya saat ini *(curr)* dan kalkulasi biaya sebelumnya *(prev)* dengan ukuran panjang string terbesar.

Setelah inisialisasi, dilakukan iterasi dan pengecekan untuk setiap karakter pada string yang lebih pendek (outer loop) terhadap karakter pada string yang lebih panjang (inner loop). Jika karakter sama, maka biaya *curr* akan sama dengan biaya *prev* (tidak ada biaya tambahan). Jika karakter berbeda, maka biaya akan ditambah satu dari minimum biaya untuk menyamakan string pertama dan string kedua, antara melalui *insert* (mengecek dari biaya sejauh ini), *remove* (mengecek dari biaya sampai karakter saat ini), dan *replace* (cek dari biaya sebelumnya). Setelah iterasi selesai, pada index terakhir dari array, akan didapatkan *cost* total yang menyatakan levenshtein distance antara kedua string. Setelah mendapatkan nilai levenshtein distance, akan dilakukan pengecekan jarak dengan threshold yang telah ditentukan setelah normalisasi, yakni 0.8 untuk pola umum dan jika

panjang kata kunci kurang dari sama dengan empat, maka threshold disesuaikan menjadi 0.75 agar pencarian lebih akurat. Jika melebihi threshold, maka akan ditambah pada hasil.

3.2.5 Pemetaan Masalah untuk Regular Expression

Pemetaan untuk regular expression pada ekstraksi *summary* dilakukan dengan pembagian informasi pada setiap section yang berisi kalimat-kalimat berdasarkan header (list info_key_regex) yang disimpan dalam suatu kamus. Kemudian pada setiap section, akan di-ekstrak informasi-informasi riwayat pekerjaan, skills, dan riwayat pendidikan dengan menggunakan notasi regex umum. Untuk riwayat pekerjaan, informasi yang diambil adalah tahun (periode), perusahaan, dan posisi yang dimiliki pelamar pada perusahaan tersebut. Sedangkan untuk pendidikan, informasi yang di ekstrak adalah tahun pendidikan, nama institusi pendidikan, dan *degree* (bidang) pendidikan yang ditempuh.

Pada setiap fungsi untuk mengekstrak informasi pada section, telah dibuat definisi pola regex yang disimpan dalam list. Pola ini didasarkan pada pengetahuan umum dan hasil percobaan yang ada pada dokumen CV selama pengembangan aplikasi. Untuk ekstraksi informasi riwayat pekerjaan dan pendidikan, setiap kata atau bagian kalimat yang cocok dengan pola regex akan dimasukkan pada variabel sementara. Jika seluruh bagian informasi sudah terisi (misal tahun, perusahaan, dan jabatan untuk ekstraksi riwayat pekerjaan), maka informasi tersebut akan disimpan sebagai informasi yang utuh. Informasi yang memiliki *missing value* atau properti tidak akan disimpan.

Kumpulan pola regex pada fungsi extract_education dirancang untuk mengekstrak informasi terkait pendidikan dari teks, terbagi dalam tiga kategori utama: gelar, institusi, dan tanggal. Pola dalam degree_patterns mencakup istilah umum seperti bachelor, phd, dan certificate, serta singkatan gelar seperti B.S., M.A., dan Ph.D., termasuk juga istilah tingkat studi seperti undergraduate dan postgraduate, serta frasa high school. Pola institution_patterns dirancang untuk menangkap kata-kata umum yang menunjukkan institusi pendidikan seperti university, college, dan academy, serta frasa seperti of Indonesia yang biasanya mengikuti nama institusi. Sementara itu, date_patterns digunakan untuk mendeteksi tahun pendidikan, baik dalam bentuk tunggal (1995, 2010), rentang tahun (1998 - 2002 atau 2015 to 2019), maupun frasa seperti graduated 2014 atau class of 2006. Gabungan pola ini memungkinkan ekstraksi data pendidikan secara efektif dari teks bebas.

Pola regex pada fungsi extract_job_history digunakan untuk mengekstrak informasi pekerjaan dari teks, terdiri dari tiga kategori: jabatan (job_title_patterns), perusahaan (company_patterns), dan tanggal (date_patterns). Pola jabatan mencakup kata kunci umum seperti manager, developer, chef, serta kombinasi level posisi dengan nama jabatan seperti Senior Engineer atau Lead Analyst. Untuk perusahaan, pola mencocokkan

istilah yang sering digunakan dalam nama perusahaan seperti Inc, Corp, Technologies, dan Group. Sementara itu, pola tanggal dirancang untuk menangkap berbagai format periode kerja, termasuk rentang tahun seperti 1995 - 2005 atau 2010 to 2020, serta format yang mencakup kata-kata seperti present atau current sebagai akhir dari periode kerja. Gabungan pola ini memungkinkan sistem mengenali pengalaman kerja seseorang secara otomatis dari teks.

Pola regex pada fungsi extract_skill dirancang untuk mengekstrak daftar keahlian dari teks dengan pendekatan yang fleksibel terhadap variasi penulisan. Pola-pola ini terbagi menjadi beberapa kategori utama berdasarkan jenis keterampilan yang umum ditemukan dalam CV, seperti bahasa pemrograman, framework, library, perangkat lunak untuk desain, soft skills, dan keterampilan *administrative*. Contoh pola yang digunakan yaitu kata kunci seperti python, java, c++, react, docker, dan git. Selain itu, fungsi ini juga mendeteksi keterampilan yang dituliskan dalam format daftar, baik yang dipisahkan dengan koma (,), simbol bullet (•), maupun tanda hubung(-).

Baris-baris yang mengandung daftar keahlian akan dipisah berdasarkan pemisah (delimiter) tersebut, lalu setiap elemennya akan dibersihkan dan dicocokkan dengan pola yang ada. Hasil pencocokan akan dikonversi ke format judul (*title case*) untuk keseragaman dan disimpan dalam daftar keterampilan akhir. Jika ada keterampilan yang tidak dikenali secara eksplisit oleh regex tetapi ditulis dalam baris pendek (kurang dari tiga kata), baris tersebut tetap akan dimasukkan sebagai kemungkinan keterampilan. Untuk menjaga kualitas hasil ekstraksi, dilakukan penghapusan duplikat dan pemotongan daftar keterampilan jika melebihi batas maksimum tertentu (misalnya 10 skill). Pendekatan ini memungkinkan sistem mengekstrak keahlian teknis dan non-teknis secara efektif dari berbagai format penulisan dalam teks bebas.

3.3 Fitur Fungsional dan Arsitektur Aplikasi

SignHire adalah program desktop recruitment yang dibangun menggunakan Python dan memanfaatkan framework CustomTkinter untuk antarmuka pengguna. Program ini menerapkan arsitektur Model-View-Controller (MVC) yang memisahkan antara logika bisnis, tampilan antarmuka, dan pengendalian alur data. Komponen model bertugas menangani algoritma pattern matching, view mengelola tampilan dan interaksi pengguna, sementara controller mengkoordinasikan komunikasi antara model dan view serta mengatur koneksi dengan database MySQL yang menyimpan data CV kandidat.

Halaman utama program SignHire menyediakan antarmuka bagi pengguna untuk melakukan pencarian kandidat berdasarkan kata kunci tertentu. Pada halaman ini, pengguna dapat memasukkan multiple keywords, memilih algoritma pattern matching yang diinginkan (KMP, Boyer-Moore, atau Aho Corassick), serta menentukan maksimal

jumlah cv yang akan ditampilkan. Setelah menerima input pencarian yang valid, backend akan menjalankan proses pencarian pattern matching secara in-memory terhadap seluruh dataset CV yang telah diekstrak.

Program ini menerapkan *dual-phase matching approach* dimana program melakukan exact matching terlebih dahulu menggunakan algoritma yang dipilih pengguna, kemudian jika exact match tidak menghasilkan kecocokan yang memadai, program akan melanjutkan pencarian dengan fuzzy matching menggunakan Levenshtein Distance. Hasil pencarian kemudian diagregasi dan diranking berdasarkan jumlah kecocokan kata kunci, dipresentasikan melalui GUI dengan informasi waktu eksekusi terpisah untuk exact match dan fuzzy match. Sebagai pelengkap, program ini juga menyediakan fitur tampilan ringkasan CV dan akses ke CV asli, memberikan gambaran menyeluruh bagi pengguna dalam proses evaluasi kandidat.

3.4 Contoh Ilustrasi Kasus

3.4.1 Ilustrasi Kasus Exact Matching: Knuth Morris Pratt

Ketika keyword diisi dengan sebuah kata, yaitu “python” dan dipilih algoritma KMP dengan jumlah CV yang diharapkan sebanyak 5, maka pemrosesan akan dilakukan dengan membuat array border function (Longest Prefix Suffix/LPS) terhadap keyword (pattern). Selanjutnya algoritma KMP akan dipanggil dengan melakukan iterasi pengecekan terhadap keseluruhan string panjang yang sudah ada dipetakan dalam array cv_database. Bila pada huruf sekarang yang dibandingkan ternyata berbeda, maka akan dilakukan pergeseran ke arah kanan yang jaraknya adalah dari kolom yang bersesuaian pada LPS. Pada contoh ilustrasi kasus ini, karena python memiliki LPS yang semua isinya 0 karena tidak ada alfabet yang sama dan prefix juga suffix yang identik, maka pengecekan akan dilakukan satu persatu tiap indexnya. Bila ditemukan kata yang sama, maka akan disimpan dalam array matches dan setiap penemuan ini akan dicari posisinya, jumlah kemunculan pada tiap CV, dan id CV. Hasil ini akan menjadi acuan untuk melakukan sorting pada hasil CV yang menentukan urutan tampil pada GUI nantinya dan penanda suatu CV sehingga dapat ditampilkan informasi summary dan CV nya.

3.4.2 Ilustrasi Kasus Fuzzy Matching: Levenshtein-Distance

Bagian akan mensimulasikan fuzzy matching dengan data pada dummy.txt yang ada pada repository, Ketika kata kunci keywords = ['kook', 'coos', 'computer'] dimasukkan, dilakukan pencocokan masing-masing kata kunci dengan setiap kata-kata yang ada pada long string dari dummy.txt. Untuk setiap pasangan kata, akan dilakukan pencocokan dengan fungsi levenshtein_calculation.

Setiap pasangan kata akan dicari masing-masing panjangnya. Panjang kata yang lebih besar akan dijadikan ukuran array untuk penyimpanan informasi biaya sementara (algoritma ini mengaplikasikan pemrograman dinamis). Informasi biaya ini adalah besar jarak levenshtein sementara. Jika salah satu string kosong, langsung dikembalikan nilai panjang string pembanding yang tidak kosong. Lalu, dikakukan inisialisasi array *prev* dan *curr*. Array *prev* berisi elemen angka sesuai dengan index. Sedangkan *curr* berisi 0. Kemudian, dilakukan iterasi pencocokan dengan outer loop string yang pendek dan inner loop string yang panjang. Jika karakter pada kedua string sama, maka tidak ada biaya tambahan. Jika berbeda, maka akan ditambahkan satu dari minimal biaya *insert, remove,* atau *replace* ke array *curr* untuk mencatat biaya (jarak levensthein) terbaru. Untuk setiap pengecekan karakter pada string pendek, akan dilakukan penyalinan nilai levenshtein distance pada array *curr* ke *prev*. Elemen terakhir pada array *curr* adalah hasil akhir perhitungan jarak levensthein untuk kedua string.

Setelah didapatkan jarak levenshtein-nya, nilai tersebut akan dibandingkan dengan threshold 0.6 setelah dinormalisasi terhadap panjang string maksimal diantara keduanya. Jika *pass* (lebih besar dari threshold) akan dimasukkan pada hasil.

3.4.5 Ilustrasi Kasus Summary Regular Expression

Bagian ini mensimulasikan proses ekstraksi informasi penting dari sebuah dokumen teks bernama regex.txt menggunakan pendekatan *regular expression* (regex). File tersebut berisi informasi tidak terstruktur seperti riwayat pekerjaan, skill, dan latar belakang pendidikan. Saat program dijalankan, tiap baris pada file akan dibaca dan dibersihkan dari spasi kosong. Program kemudian menelusuri setiap baris dan mencocokkan dengan daftar pattern kata kunci yang telah ditentukan, seperti skill, education, dan experience, untuk mengelompokkan baris-baris tersebut ke dalam kategori informasi yang relevan.

Proses pencocokan ini menggunakan ekspresi reguler (regex) untuk mengenali apakah sebuah baris merupakan bagian dari suatu kategori. Jika cocok, maka baris tersebut dimasukkan ke dalam kelompok yang sesuai. Misalnya, jika ditemukan baris seperti "Education" atau "Skills", maka baris-baris setelahnya dianggap sebagai konten dari kategori tersebut hingga ditemukan kata kunci kategori baru. Setelah semua kelompok terbentuk, akan dilakukan proses ekstraksi informasi lebih lanjut untuk tiga kelompok utama: pendidikan (*education*), riwayat kerja (*experience*), dan keahlian (*skill*).

Untuk bagian pendidikan, akan dilakukan pencarian pola yang umum seperti gelar (misalnya Bachelor, Master, PhD), institusi (misalnya University, College), dan rentang tahun atau tahun kelulusan. Untuk setiap baris yang memuat informasi-informasi ini, akan dibentuk entri berupa dictionary yang mencakup degree, institution, dan year.

Sementara itu, ekstraksi riwayat kerja menggunakan ekspresi reguler untuk mencari jabatan (misalnya Manager, Engineer, Chef), nama perusahaan (misalnya Company, Group, Technologies), dan periode kerja (misalnya 2015 - 2020). Baris-baris pendek dengan format seperti itu dianggap memuat informasi pekerjaan dan akan dimasukkan ke dalam daftar riwayat kerja.

Untuk keahlian, pendekatan yang digunakan mengandalkan pencocokan terhadap kata-kata kunci teknologi dan keterampilan populer, seperti bahasa pemrograman (Python, Java), framework (React, Django), platform (AWS, Linux), serta soft skill seperti Team Leadership atau Problem Solving. Pola-pola ini memungkinkan program mengekstrak daftar keahlian dari baris-baris yang panjang maupun daftar bullet points atau dipisahkan dengan koma.

Setelah seluruh proses selesai, hasilnya berupa ringkasan data terstruktur dari tiga kategori utama: education, experience, dan skill, yang siap digunakan untuk keperluan analisis lebih lanjut, seperti membandingkan kandidat CV dengan lowongan pekerjaan menggunakan teknik pencocokan berbasis teks.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Struktur Kode Program

Pada program ini, sebagian kode yang krusial dibuat dengan paradigma berorientasi objek (OOP) untuk mempermudah pemrosesan metode atau fungsi yang merupakan tanggung jawab masing-masing kelas. Namun, terdapat beberapa kelas yang tidak menggunakan paradigma ini karena proses lebih bersifat prosedural. Folder program disusun sebagai berikut:

```
TUCIL3_13523050_13523079
├── data/                          # Folder untuk file hasil kompilasi
├── doc/
│   └── laporan.pdf               # Laporan tugas dalam format PDF
├── src/                          # Folder source code utama
│   ├── controller/
│   │   ├── cv_data_manager.py
│   │   ├── db_setup.py
│   │   └── seeder.py
│   ├── gui/
│   │   ├── assets/
│   │   ├── components/
│   │   │   ├── about_page.py
│   │   │   ├── cv_summary_window.py
│   │   │   ├── developer_page.py
│   │   │   ├── home_page.py
│   │   │   ├── main_window.py
│   │   │   └── splash_screen.py
│   ├── model/
│   │   ├── aho_corassick.py
│   │   ├── boyer_moore.py
│   │   ├── encryptor.py
│   │   ├── knuth_morris_pratt.py
│   │   ├── levenshtein_distance.py
│   │   └── regex.py
│   └── app.py
├── README.md                     # Dokumentasi utama proyek
└── run.bat                       # Tool untuk compile dan run program
```

4.2 Fungsi dan Prosedur

4.2.1 Knuth Morris Pratt

Nama Fungsi atau Prosedur	Deskripsi Singkat
knuth_morris_pratt(data:str, keyword:list) → dict	Mengembalikan jumlah kemunculan suatu kata kunci menggunakan algoritma KMP

	dalam bentuk dictionary.
knuth_morris_pratt_with_cv_info(cv_database: dict, keyword: list) → dict	Mencari kata kunci di database CV dan memberikan informasi lengkap termasuk skor, posisi, dan ranking CV berdasarkan jumlah kecocokan.
search_cvs_with_details(cv_database: dict, keywords: list, top_n: int = 5) → list	Mengambil top N CV terbaik dari hasil pencarian dan memberikan detail lengkap tentang kecocokan kata kunci di setiap CV.
build_bf_array(pattern: str) → list()	Membangun array border function yang digunakan dalam algoritma KMP untuk menentukan lompatan saat terjadi ketidakcocokan.
kmp_search()	Implementasi algoritma Knuth-Morris-Pratt untuk mencari semua posisi kemunculan pattern dalam teks dengan efisien

4.2.2 Boyer Moore

Nama Fungsi atau Prosedur	Deskripsi Singkat
build_last_occurrence(pattern: str) → dict	Membuat dictionary yang menyimpan posisi terakhir setiap karakter dalam pattern untuk digunakan dalam algoritma Boyer-Moore.
boyer_moore_search(text: str, pattern: str) → list	Implementasi algoritma Boyer-Moore untuk mencari semua posisi kemunculan pattern dalam teks dengan melompat berdasarkan karakter yang tidak cocok.
boyer_moore_search_keyword_list(text: str, keywords: list) → dict	Mencari berapa kali setiap kata kunci muncul dalam teks menggunakan algoritma Boyer-Moore dan mengembalikan hasil dalam bentuk dictionary.
boyer_moore_with_cv_info(cv_database: dict, keywords: list) → dict	Mencari kata kunci di database CV menggunakan algoritma Boyer-Moore dan memberikan informasi lengkap termasuk

	skor, posisi, dan ranking CV.
search_cvs_boyer_moore(cv_database: dict, keywords: list, top_n: int = 5) → list	Mengambil top N CV terbaik dari hasil pencarian Boyer-Moore dan memberikan detail lengkap tentang kecocokan kata kunci di setiap CV.

4.2.2 Aho Corassick

Nama Fungsi atau Prosedur	Deskripsi Singkat
add_keyword(self, keyword)	Menambahkan kata kunci ke dalam struktur trie dengan menandai akhir kata menggunakan simbol '$'
build_failure_function(self)	Membangun failure function dan output function untuk automaton Aho-Corasick menggunakan BFS
_get_node_by_id(self, node_id)	Helper function untuk mencari dan mengembalikan node dalam trie berdasarkan ID
_get_matches_at_node(self, node)	Mengambil semua pattern yang cocok pada node tertentu termasuk dari output function
_reconstruct_pattern(self, target_node)	Merekonstruksi pattern lengkap dari root ke target node menggunakan DFS
search(self, text)	Melakukan pencarian semua keywords dalam teks menggunakan automaton Aho-Corasick dan mengembalikan jumlah kemunculan setiap keyword
aho_corasick_search(text: str, keywords: list) → dict	Wrapper function untuk mencari keywords dalam teks menggunakan Aho-Corasick dengan normalisasi case-insensitive.
aho_corasick_search_with_cv_info(cv_database: dict, keywords: list) → dict	Mencari keywords di database CV menggunakan Aho-Corasick dan memberikan informasi lengkap termasuk skor dan posisi
search_cvs_with_aho_corasick(cv_database	Mengambil top N CV terbaik dari hasil

: dict, keywords: list, top_n: int = 5) → list	pencarian Aho-Corasick dengan detail lengkap kecocokan keywords

4.2.3 Levenshtein Distance

Nama Fungsi atau Prosedur	Deskripsi Singkat
levenshtein_distance(data: list, keyword: list) → dict	Menghitung berapa banyak kata dalam data yang mirip dengan setiap keyword berdasarkan similarity threshold menggunakan Levenshtein distance
is_pass(string1: str, string2: str, dist: int) → bool	Menentukan apakah dua string dianggap mirip dengan membandingkan similarity ratio mereka terhadap threshold yang ditentukan
levenshtein_calculation(string1: str, string2: str) → int	Menghitung jarak Levenshtein antara dua string menggunakan dynamic programming untuk menentukan minimum operasi insert, delete, atau replace
levenshtein_search_cv(cv_content: str, keywords: list) → dict	Membersihkan teks CV menjadi kata-kata alphanumeric dan mencari kecocokan fuzzy dengan keywords menggunakan Levenshtein distance
levenshtein_search_with_cv_info(cv_database: dict, keywords: list) → dict	Mencari keywords di database CV menggunakan fuzzy matching Levenshtein dan memberikan informasi lengkap termasuk skor, posisi, dan ranking CV
search_cvs_with_levenshtein(cv_database: dict, keywords: list, top_n: int = 5) → list	Mengambil top N CV terbaik dari hasil pencarian Levenshtein dengan detail lengkap kecocokan keywords yang mirip

4.2.4 Regex

Nama Fungsi atau Prosedur	Deskripsi Singkat
count_words(s: str) -> int	Menghitung jumlah kata dalam sebuah string dengan memisahkan berdasarkan

	spasi
extract_information_group(filename : str) -> dict	Membaca file CV dan mengekstrak informasi ke dalam grup-grup (skill, experience, education)
generate_summary(data : dict) -> dict	Memproses setiap grup informasi untuk menghasilkan summary terstruktur.
extract_education(data : list) -> list	Mengekstrak informasi pendidikan dari teks menggunakan regex pattern untuk mencari degree, institusi, dan tahun
extract_job_history(data : list) -> list	Mengekstrak riwayat pekerjaan dari teks menggunakan regex pattern untuk mencari role, company, dan periode kerja
extract_skill(data : list) -> list	Mengekstrak skills dari teks menggunakan regex pattern untuk teknologi, tools, dan soft skills

4.2.5 Encryptor

Nama Fungsi atau Prosedur	Deskripsi Singkat
_shift_char(self, c, k, encrypt=True)	Menggeser karakter berdasarkan nilai key menggunakan modular arithmetic pada rentang karakter ASCII printable (32-126).
_apply_cipher(self, text: str, encrypt=True)	Menggeser setiap karakter dalam teks menggunakan karakter key yang berulang
encrypt(self, plaintext: str) → str	Mengenkripsi plaintext menggunakan cipher kemudian mengenkode hasilnya dengan base64 URL-safe encoding
decrypt(self, ciphertext: str) → str	Mendekripsi ciphertext dengan mendecode base64 terlebih dahulu kemudian menerapkan cipher terbalik untuk mendapatkan plaintext asli

4.2.6 CV Data Manager

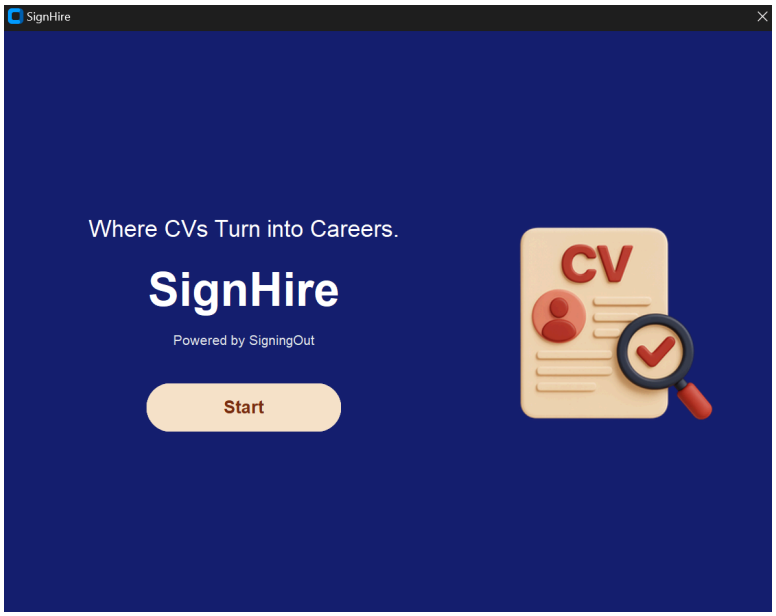
Nama Fungsi atau Prosedur	Deskripsi Singkat
get_cv_paths(self) -> dict	Mengambil semua path file CV dari database ApplicationDetail dan mengembalikannya dalam bentuk dictionary dengan format cv_id sebagai key
get_applicant_data(self, detail_ids: list) -> dict	Mengambil data profil pelamar dari database berdasarkan detail_ids dan mendekripsi field-field yang terenkripsi seperti nama, email, alamat, dan nomor telepon
extract_cv_content(self, cv_path: str, use_regex: bool = False) -> str	Mengekstrak konten teks dari file PDF CV dengan mencari path file yang valid dan menggunakan fungsi ekstraksi CV
get_cv_database_for_search(self, use_regex: bool = False) -> dict	Membuat database CV lengkap untuk pencarian dengan mengekstrak konten semua CV dan menyimpannya dalam cache untuk efisiensi.
get_applicant_summary_data(self, detail_id: int) -> dict	Mengambil data lengkap pelamar termasuk informasi personal dan hasil ekstraksi CV (skills, pengalaman kerja, pendidikan) untuk keperluan summary
extract_cv_content_and_process(self, cv_path: str, use_regex: bool = False) -> dict	Mengekstrak konten CV dan memproses dengan regex untuk mendapatkan informasi terstruktur seperti skills, experience, dan education
get_default_job_history(self) -> list	Memberikan data default pengalaman kerja ketika ekstraksi CV gagal atau tidak tersedia
get_default_education(self) -> list	Memberikan data default pendidikan ketika ekstraksi CV gagal atau tidak tersedia
get_cv_file_path(self, detail_id: int) -> str	Mengambil path file CV berdasarkan detail_id tertentu dari database
get_fallback_summary_data(self) -> dict	Memberikan data fallback lengkap

	dengan nilai default ketika pengambilan data applicant gagal

4.3 Tata Cara Penggunaan Program

4.3.1 Landing Page

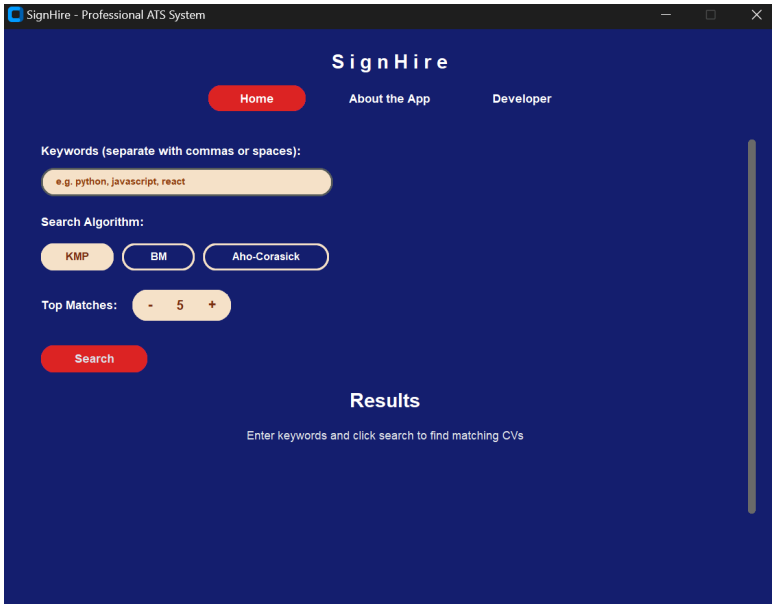
Pada program SignHire, saat pertama kali dijalankan, pengguna akan disambut dengan landing page sebagai tampilan awal. Halaman ini menampilkan informasi berupa nama program, tagline, serta nama tim pengembang, yaitu SigningOut. Setelah itu, pengguna dapat menekan tombol "Start" untuk melanjutkan dan diarahkan ke halaman utama (home page) dari program.



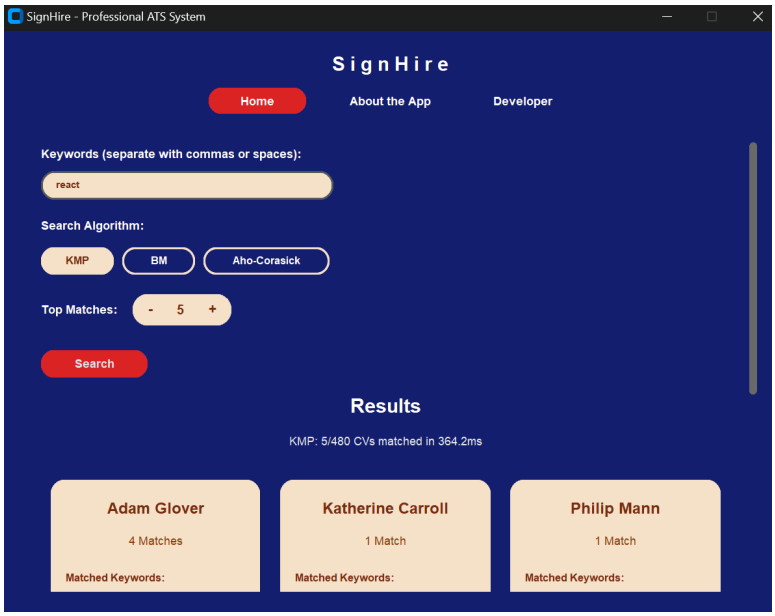
Gambar 10. Landing Page
Sumber: Dokumentasi Pribadi

4.3.2 Home Page

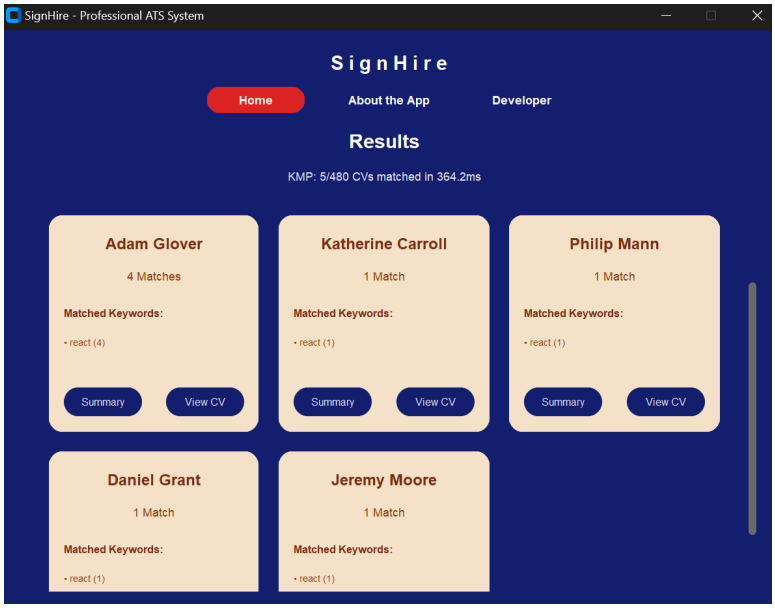
Halaman Home Page menampilkan sebuah kotak input untuk memasukkan kata kunci (*keywords*), dilengkapi dengan tombol untuk memilih algoritma pencocokan yang akan digunakan serta menentukan jumlah CV yang ingin ditampilkan. Setelah pengguna menekan tombol "Search", hasil pencarian akan ditampilkan dalam bentuk kartu (card) sebanyak jumlah yang diminta.



Gambar 11. Home Page 1
Sumber: Dokumentasi Prbadi



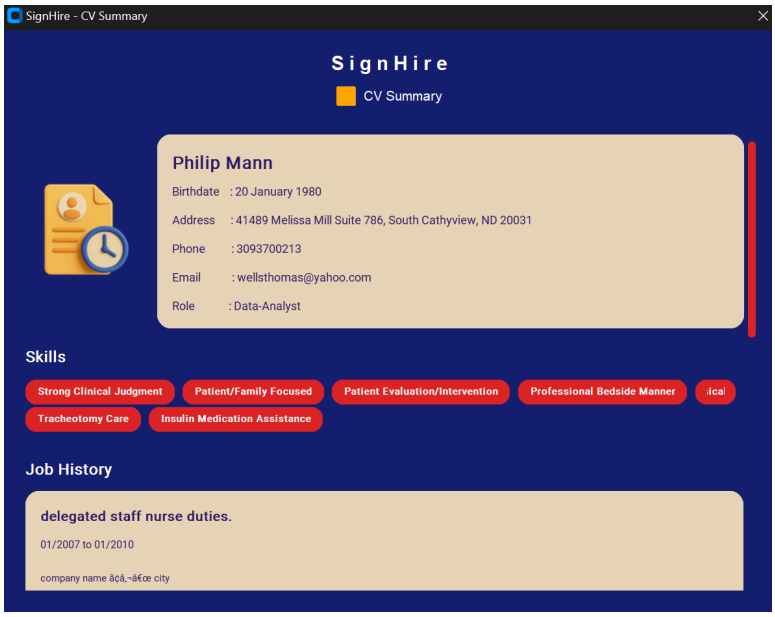
Gambar 12. Landing Page 2
Sumber Dokumentasi Pribadi



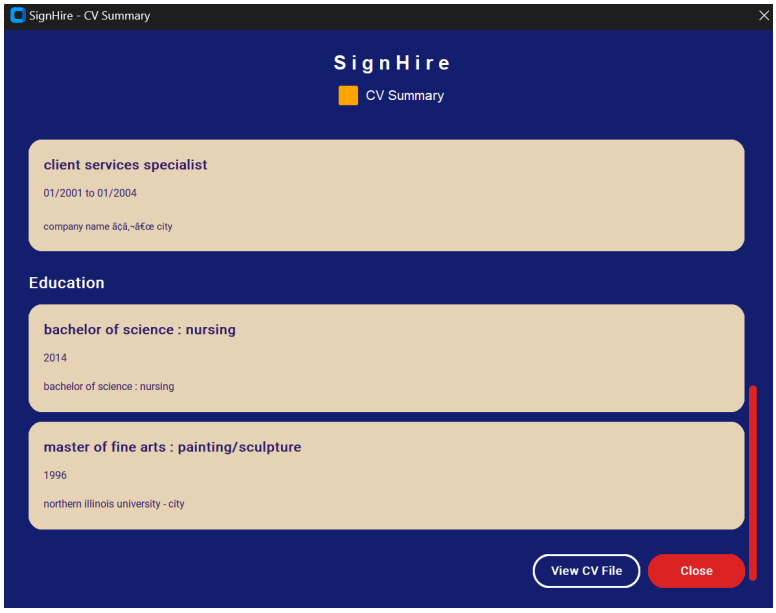
Gambar 13. Landing Page 3
Sumber: Dokumentasi Pribadi

4.3.3 Summary Page

Saat pengguna menekan tombol Summary pada kartu hasil pencarian, sistem akan menampilkan halaman Summary Page yang berisi ringkasan informasi dari CV terkait, seperti tanggal lahir, alamat, nomor telepon, email, peran atau posisi, keterampilan, riwayat pekerjaan, serta riwayat pendidikan.



Gambar 14. Summary Page 1
Sumber: Dokumentasi Pribadi



Gambar 15. Summary Page 2
Sumber: Dokumentasi Pribadi

4.3.4 View CV Page

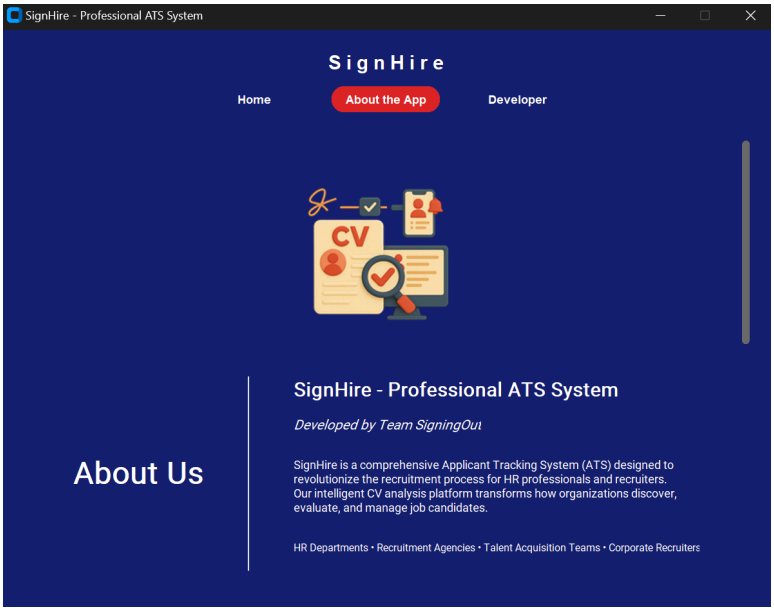
Saat pengguna menekan tombol View CV pada kartu hasil pencarian maupun pada halaman Summary, sistem akan menampilkan file PDF dari CV terkait. Khusus pada bagian kartu hasil pencarian, sistem juga menampilkan isi hasil ekstraksi dari file PDF tersebut.

Gambar 16. View CV Page

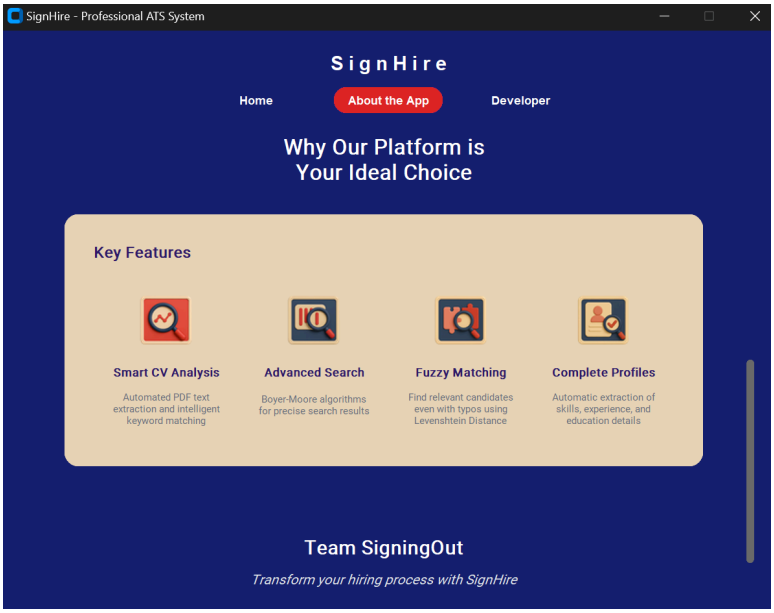
Sumber: Dokumentasi Pribadi

4.3.5 About the App Menu

Salah satu menu yang dapat dipilih oleh pengguna adalah About the App. Pada halaman ini, ditampilkan penjelasan singkat mengenai program serta berbagai fitur yang tersedia di dalamnya.



Gambar 17. About the App Page 1
Sumber: Dokumentasi Pribadi

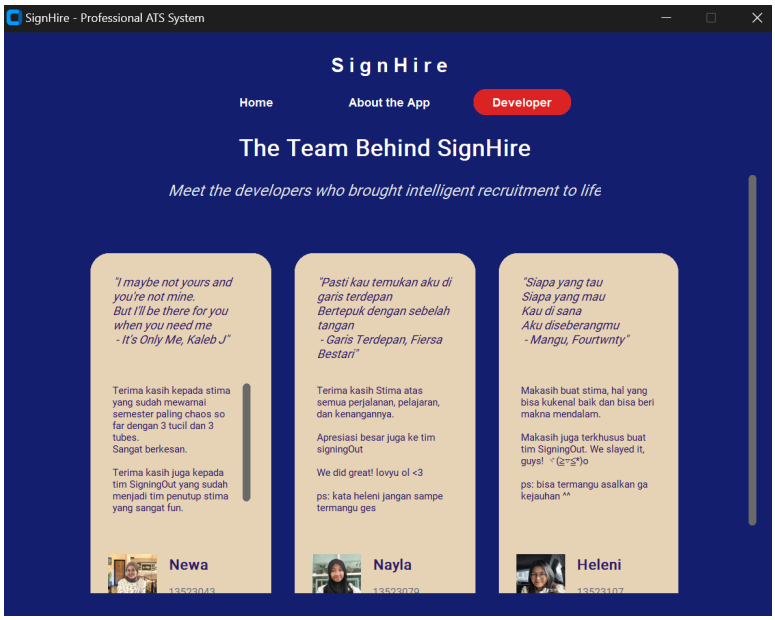


Gambar 18. About the App Page 2

Sumber: Dokumentasi Pribadi

4.3.6 Developer Menu

Salah satu menu yang dapat dipilih oleh pengguna adalah Developer. Halaman ini menampilkan profil para pengembang di balik pembuatan program SignHire, yang mencakup foto, nama, NIM, serta kutipan atau pesan singkat dari masing-masing anggota tim.

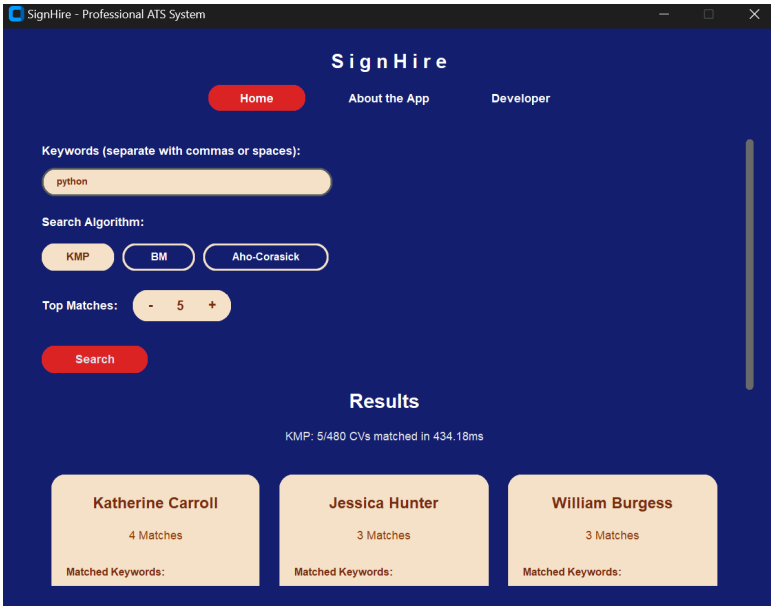


Gambar 19. Developer Menu Page
Sumber: Dokumentasi Pribadi

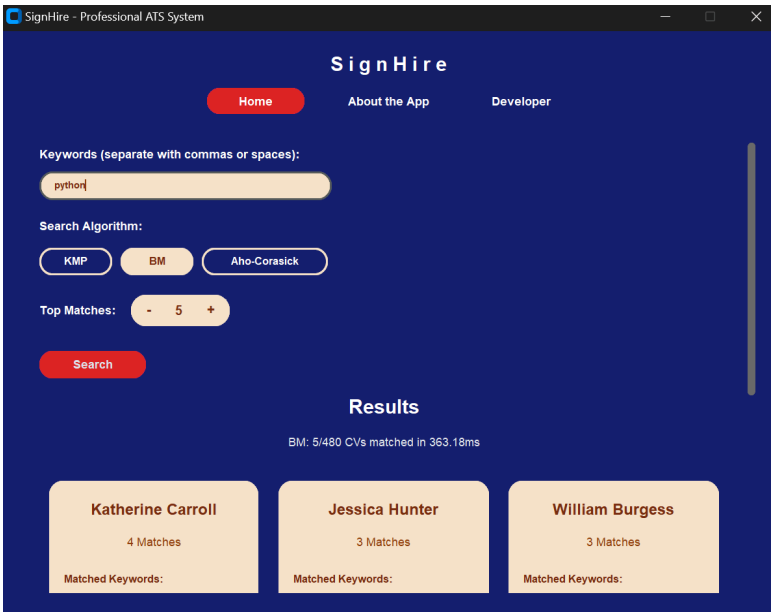
4.4 Hasil Pengujian dan Analisis

4.4.1. Test Case 1

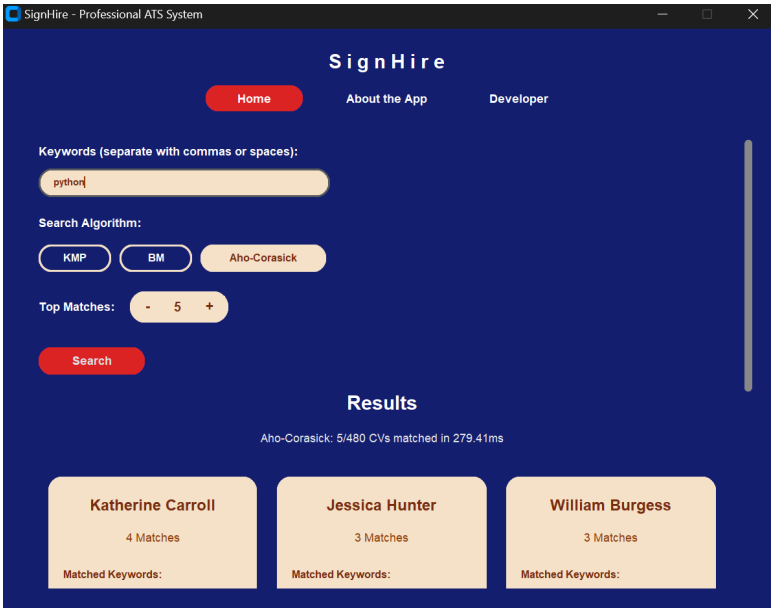
Kondisi: Pencarian dengan Exact String Matching dan jumlah CV Output Sesuai dengan Input
Input Keyword: python Top Matches: 5
Algoritma Knuth Morris Pratt



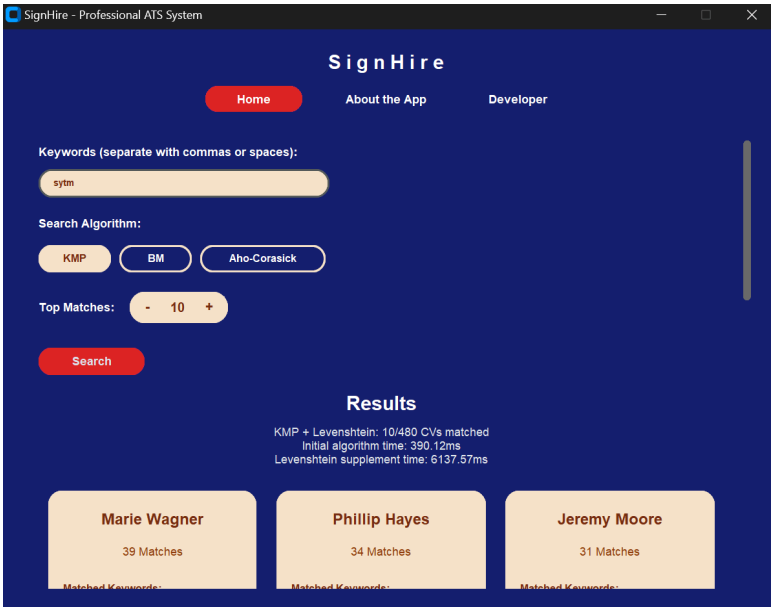
Algoritma Boyer Moore

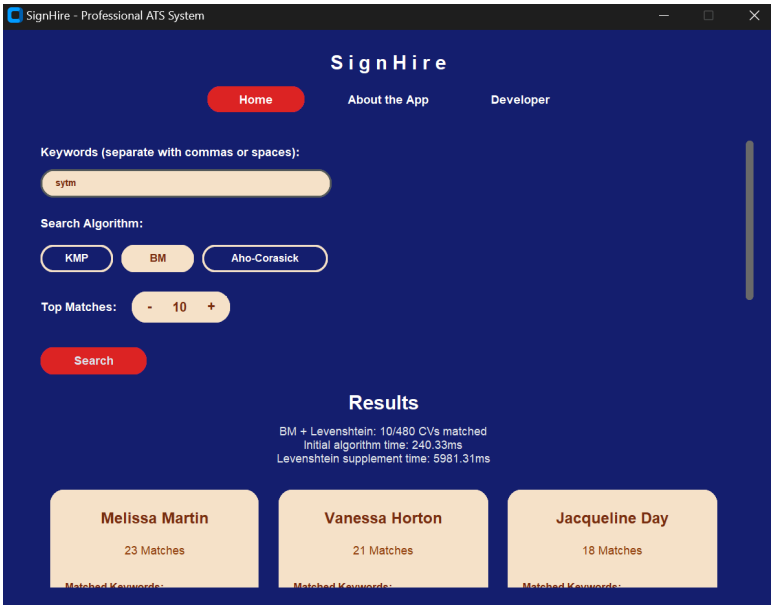
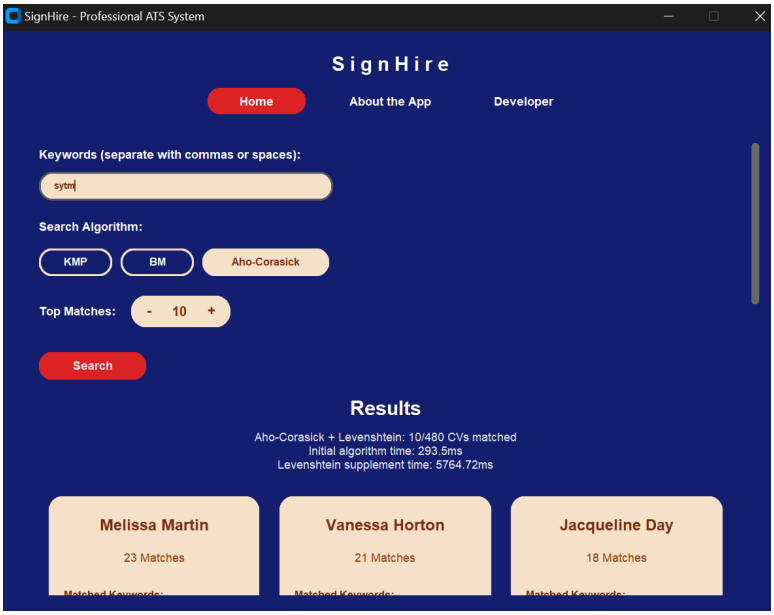


Algoritma Aho Corasick



4.4.2. Test Case 2

Kondisi: Pencarian dengan Keyword Typo
Input Keyword: sytm Top Matches: 10
Algoritma Knuth Morris Pratt → Levenshtein Distance


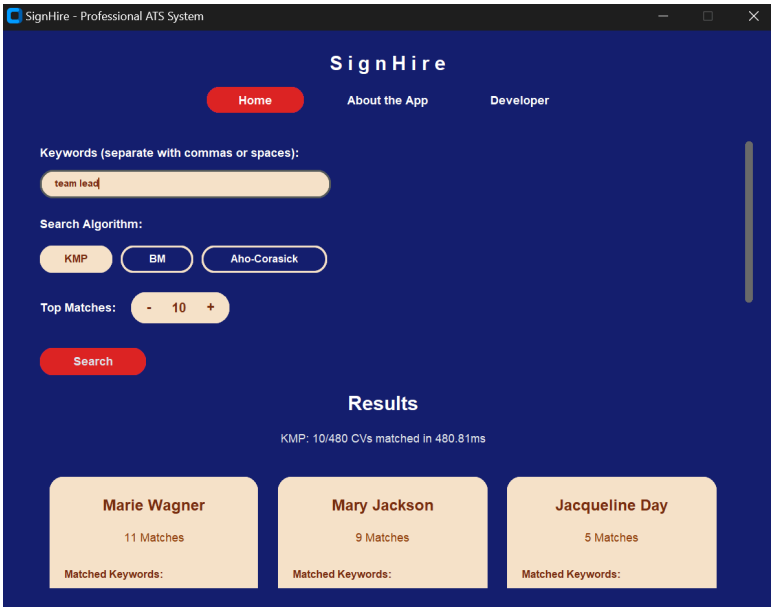
Algoritma Boyer Moore → Levenshtein Distance

Algoritma Aho-Corasick → Levenshtein Distance


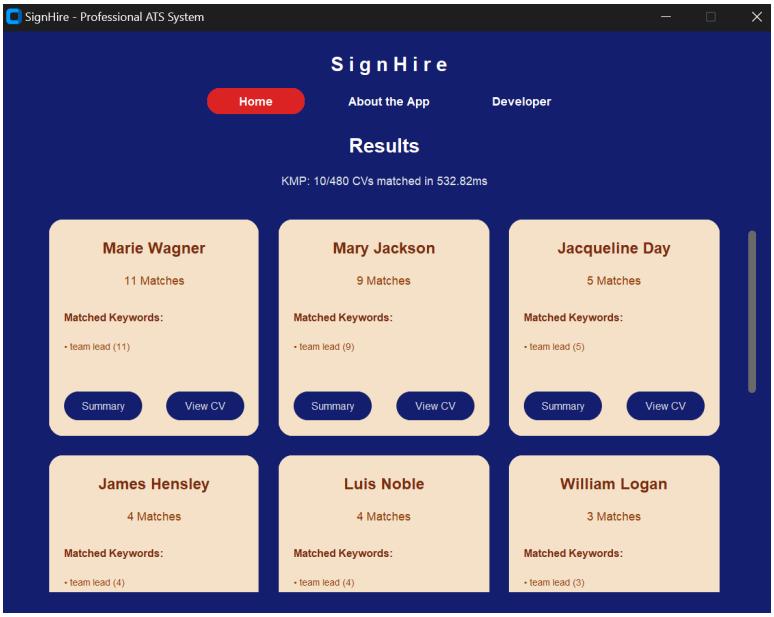
4.4.3. Test Case 3

Kondisi: Pencarian dengan 2 Kata (terdapat spasi)
Input

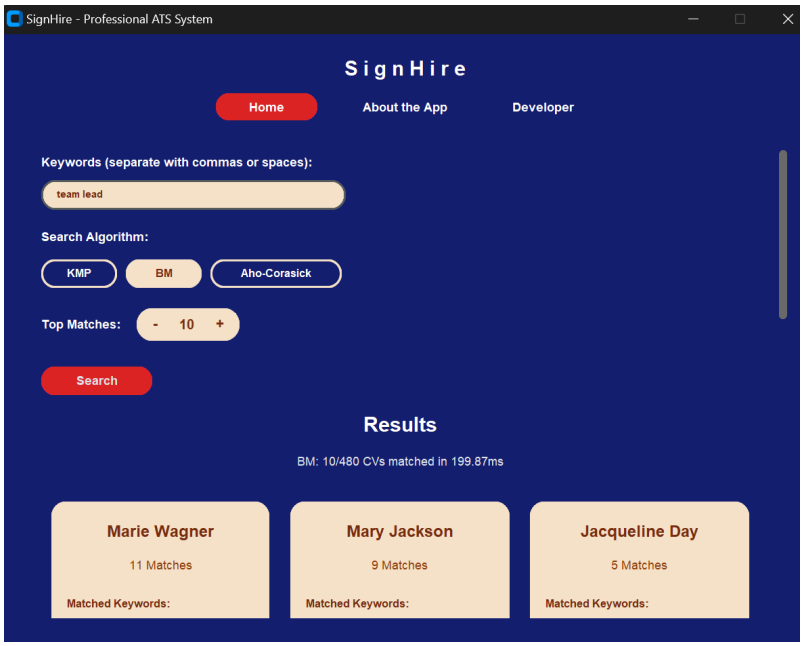
Keyword: team lead
Top Matches: 10

Algoritma Knuth Morris Pratt

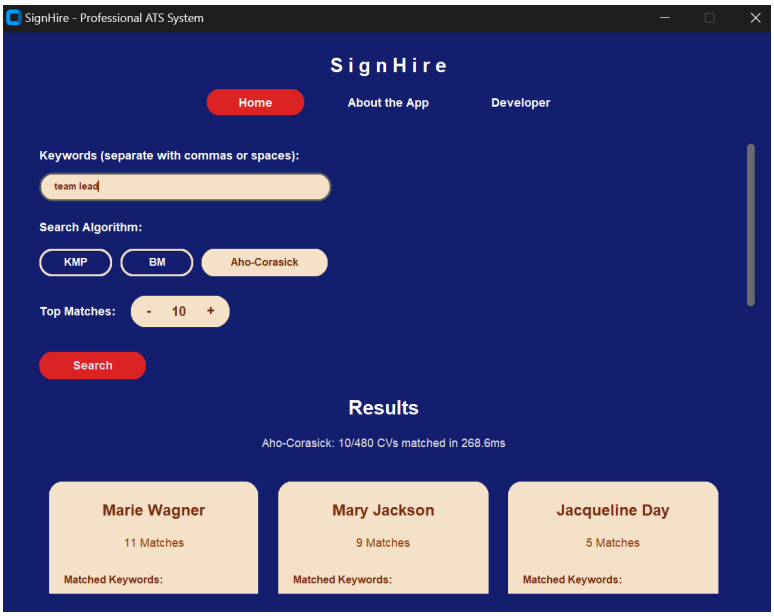




Algoritma Boyer Moore



Algoritma Aho-Corasick

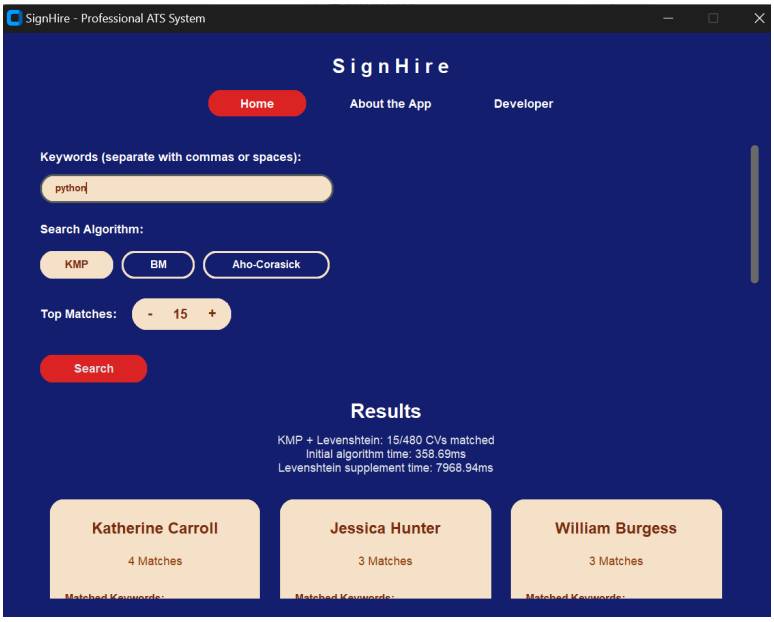


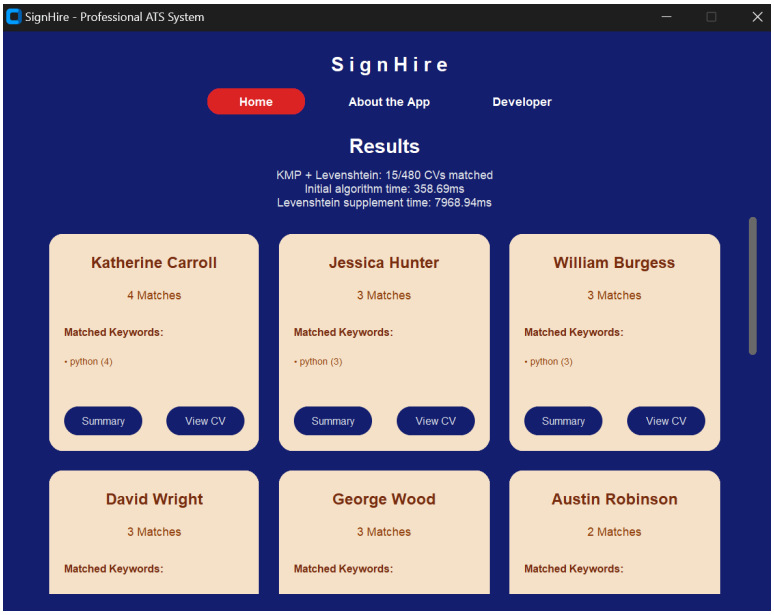
4.4.4. Test Case 4

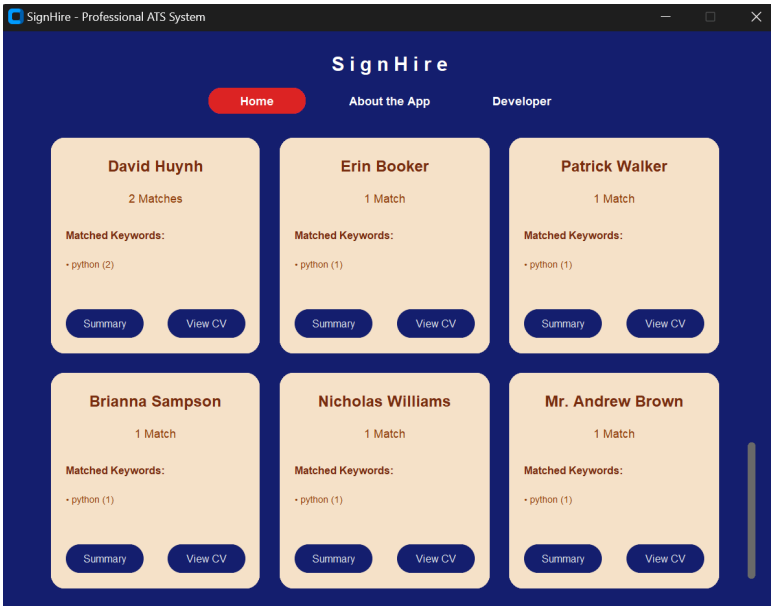
Kondisi: Pencarian dengan Jumlah CV yang Diambil Besar Sehingga Membutuhkan Levenshtein Distance

Input
Keyword: Python
Top Matches: 15

Algoritma Knuth Morris Pratt

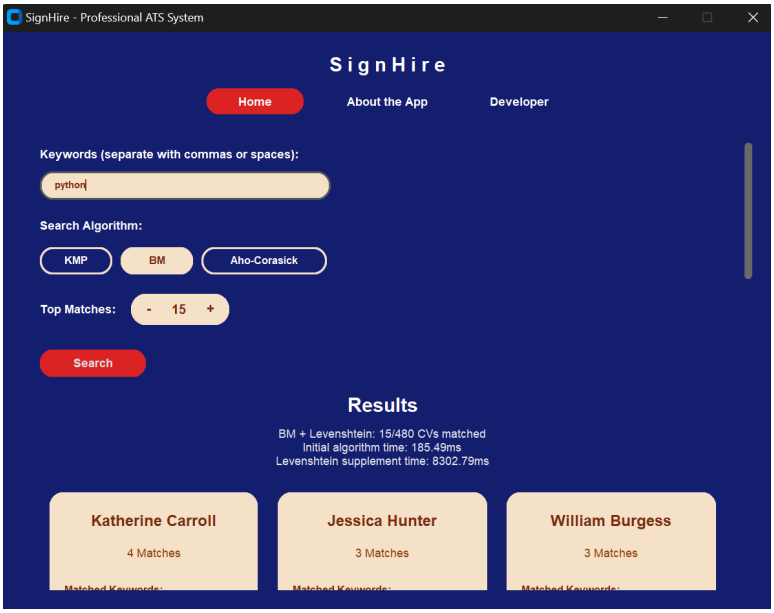




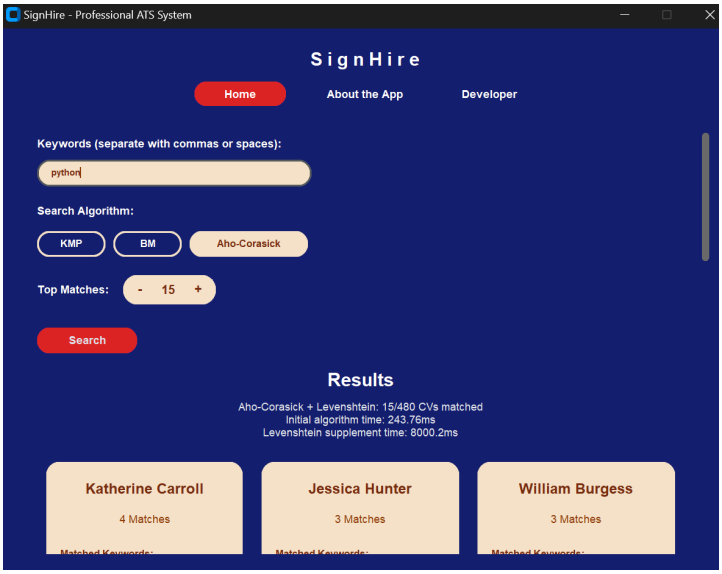


(card bagian bawah diperoleh dari Levenshtein Distance)

Algoritma Boyer Moore

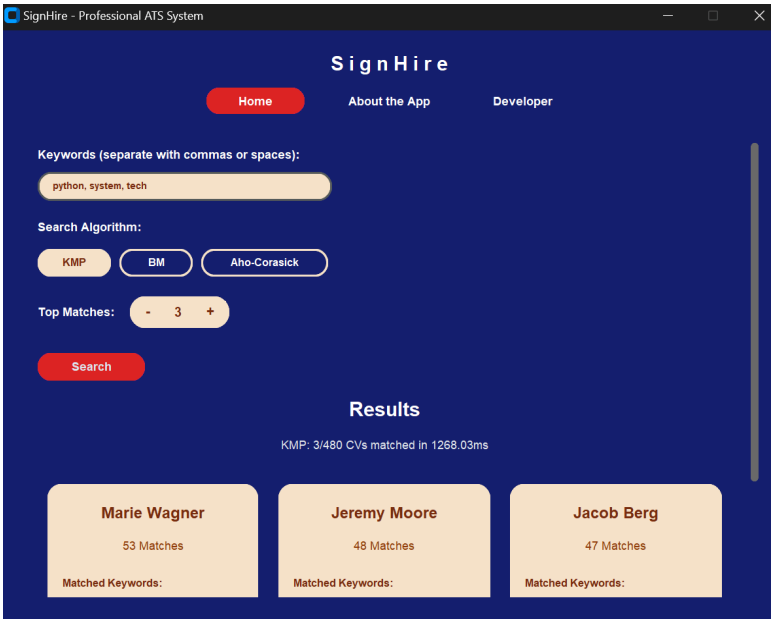


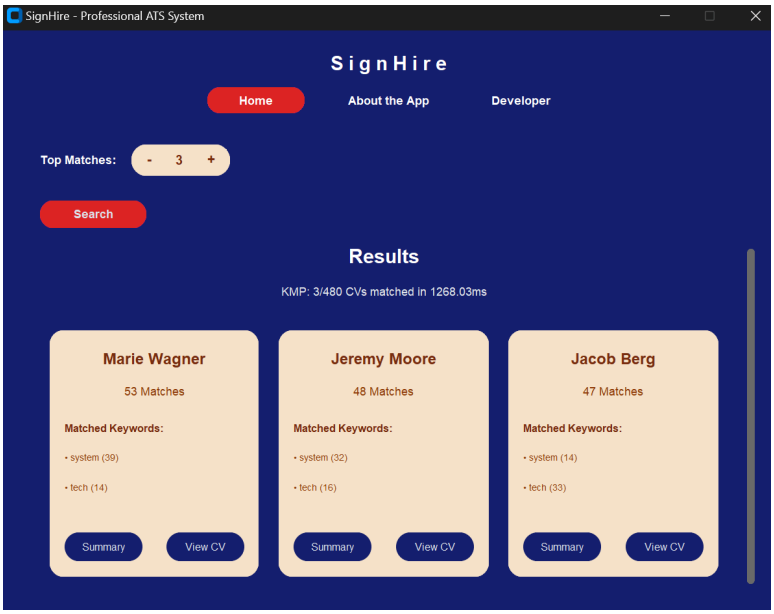
Algoritma Aho-Corasick



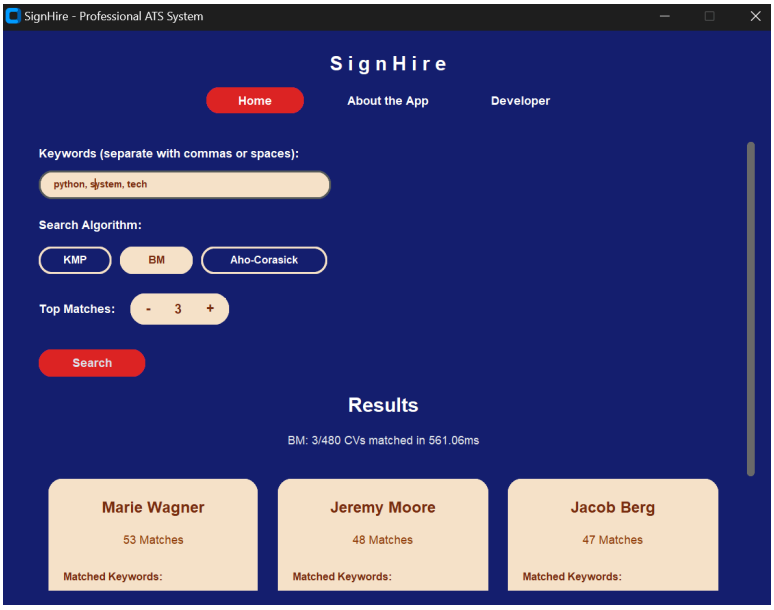
4.4.5. Test Case 5

Kondisi: Pencarian dengan lebih dari 2 kata kunci (memakai koma)
Input Keyword: Python Top Matches: 15
Algoritma Knuth Morris Pratt

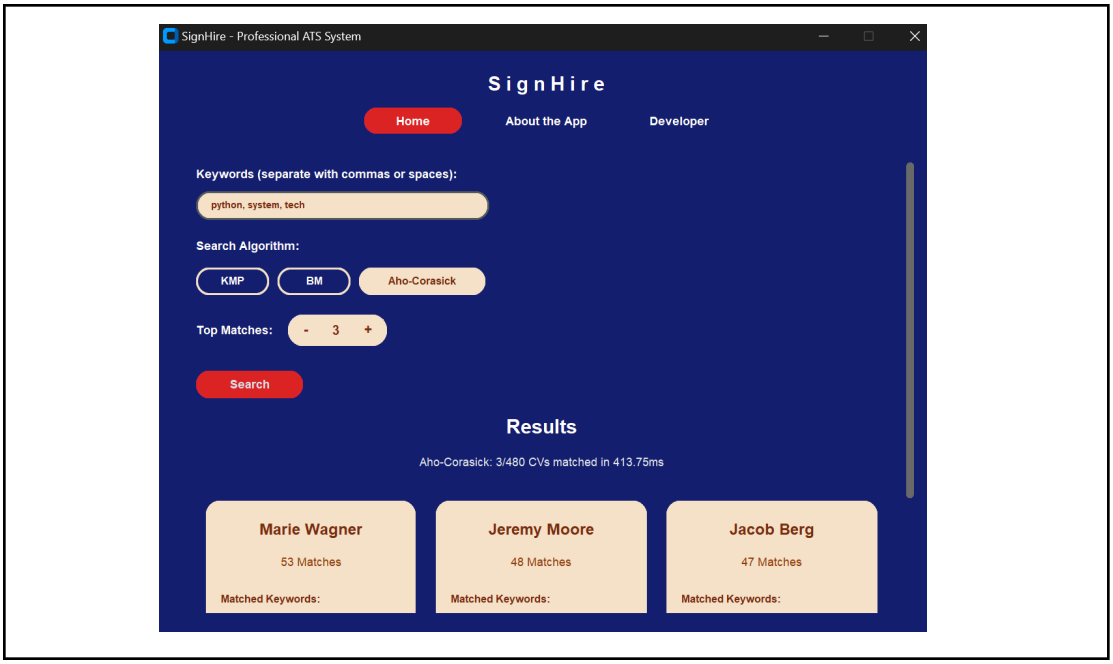




Algoritma Boyer Moore



Algoritma Aho-Corasick



4.7.7 Analisis Hasil Pengujian

Berdasarkan hasil pengujian terhadap beberapa algoritma pencocokan string yang digunakan dalam program ini, yaitu Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick (AC), dapat disimpulkan bahwa secara umum algoritma Boyer-Moore menunjukkan performa paling optimal dari segi kecepatan pencarian. Hal ini disebabkan oleh keunggulan Boyer-Moore dalam memanfaatkan informasi dari karakter teks secara efisien, sehingga lebih efektif ketika diterapkan pada teks yang panjang dan memiliki keragaman alfabet tinggi. Meskipun demikian, keunggulan ini tidak berlaku secara mutlak. Pada kondisi tertentu, seperti ketika jumlah kata kunci yang dicari sangat banyak, atau ketika struktur teks memiliki pola yang berulang, maka algoritma Aho-Corasick dapat menunjukkan performa yang lebih unggul. Aho-Corasick sangat efektif dalam menangani pencarian *multi-pattern* karena kemampuannya membangun automaton yang memungkinkan pencocokan banyak kata kunci sekaligus dalam satu kali lintasan teks.

Proses pencocokan dalam program ini terdiri dari beberapa tahapan. Pertama, seluruh file CV dalam format PDF dimuat ke dalam database SQL sebagai data terstruktur. Kemudian, setiap file diekstrak menjadi representasi string panjang menggunakan library pemrosesan PDF, agar mempermudah proses pencarian. Selanjutnya, kata kunci yang dimasukkan pengguna dicocokkan terhadap string hasil ekstraksi menggunakan algoritma yang dipilih, yaitu KMP, BM, atau AC. Ketiga algoritma ini memiliki logika pencocokan yang konsisten sehingga memberikan hasil yang sama secara kuantitas, namun berbeda dalam hal waktu pemrosesan.

Apabila tidak ditemukan kecocokan kata secara langsung, sistem secara otomatis menerapkan pencarian berbasis kemiripan menggunakan algoritma Levenshtein Distance. Pendekatan ini memungkinkan identifikasi kata-kata yang hampir mirip dengan kata kunci pengguna, sehingga kesalahan pengetikan atau variasi bentuk kata tetap dapat dikenali. Pada implementasi program ini, digunakan ambang batas (*threshold*) sebesar 0.6, yang berarti kata akan dianggap relevan jika memiliki tingkat kemiripan minimal 60% terhadap kata kunci setelah dilakukan normalisasi.

Secara keseluruhan, hasil pengujian menunjukkan bahwa pemilihan algoritma pencocokan string memiliki dampak signifikan terhadap efisiensi sistem. Boyer-Moore secara umum lebih cepat untuk skenario pencarian tunggal, namun program tetap dirancang fleksibel dengan memberikan pilihan algoritma kepada pengguna, sehingga bisa disesuaikan dengan kebutuhan dan karakteristik data yang sedang dianalisis.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada program "SignHire", kami telah berhasil mengimplementasikan algoritma KMP, Boyer Moore, dan Aho Corasick untuk menyelesaikan permasalahan pattern matching pada pencarian CV berdasarkan kata kunci. Selain itu, kami juga telah mengimplementasikan pencarian fuzzy matching menggunakan levenshtein distance sebagai fallback jika tidak ditemukannya kecocokan menggunakan exact matching sebelumnya.

Berdasarkan hasil pengujian yang telah dilakukan, dapat disimpulkan bahwa implementasi ketiga algoritma tersebut memberikan hasil yang sangat memuaskan dalam konteks aplikasi pencarian CV. Algoritma KMP menunjukkan performa yang konsisten dengan kompleksitas waktu O(n+m) dan sangat efektif untuk pencarian pola tunggal pada dokumen CV yang panjang. Sementara itu, algoritma Boyer Moore memberikan performa terbaik pada keyword dengan alfabet yang lebih beragam dan teks yang relatif panjang. Di sisi lain, algoritma Aho Corasick terbukti sangat efisien untuk pencarian multiple pattern secara simultan, yang memungkinkan pencarian beberapa kata kunci sekaligus dalam satu kali proses.

5.2 Saran

Berikut beberapa saran oleh kelompok SigningOut yang berguna untuk memaksimalkan potensi pengembangan selanjutnya:

1. Mengoptimalkan pola regex untuk meningkatkan keakuratan summary dari cv
2. Mengintegrasikan teknologi machine learning dan natural language processing untuk meningkatkan akurasi semantic matching

5.3 Refleksi

Terdapat beberapa refleksi terhadap kelompok SigningOut yang bisa menjadi bahan evaluasi untuk perbaikan kedepannnya.

1. Pengoptimalan alokasi waktu dalam pengerjaannnya untuk meningkatkan hasil program.
2. Mengurangi waktu begadang sehingga akan sangat bahagia dan ceria dalam pengerjaan tugas besar ini.

LAMPIRAN

- Tautan Repository GitHub :

  https://github.com/naylzhra/Tubes3_SigningOut

- Tautan Video

  https://youtu.be/8lTgnmGUqjQ

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

Devopedia. (n.d.). *Levenshtein Distance*. Retrieved June 14, 2025, from https://devopedia.org/levenshtein-distance

GeeksforGeeks. (n.d.). *Introduction to Levenshtein Distance*. Retrieved June 14, 2025, from https://www.geeksforgeeks.org/dsa/introduction-to-levenshtein-distance/

GeeksforGeeks. (n.d.). *Trie | (Insert and Search)*. Retrieved June 14, 2025, from https://www.geeksforgeeks.org/dsa/trie-insert-and-search/

Jain, S. (2025, February 25). *KMP Algorithm for Pattern Searching*. GeeksforGeeks. Retrieved June 14, 2025, from https://www.geeksforgeeks.org/dsa/kmp-algorithm-for-pattern-searching/

Jain, S. (2025, April 11). *Boyer Moore Algorithm for Pattern Searching*. GeeksforGeeks. Retrieved June 14, 2025, from https://www.geeksforgeeks.org/dsa/boyer-moore-algorithm-for-pattern-searching/

Munir, R. (n.d.). *Pencocokan String dengan Regular Expression (Regex)*. Retrieved 6 14, 2025, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf

Munir, R. (n.d.). *Pencocokan String (String Matching) dengan Algoritma Brute Force, KMP, Boyer Moore*. Retrieved 6 14, 2025, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf

ScienceDirect. (n.d.). *Levenshtein Distance*. Retrieved June 14, 2025, from https://www.sciencedirect.com/topics/computer-science/levenshtein-distance