Penerapan Algoritma Aho-Corasick pada Applicant Tracking System untuk Otomasi Screening Kata Kunci Kompetensi pada CV

Mahmudia Kimdaro Amin - 13524083
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: mkaxydaro@gmail.com , 13524083@std.stei.itb.ac.id

***Abstract*— Proses screening Curriculum Vitae (CV) merupakan salah satu tahap penting dalam sistem rekrutmen modern yang sering kali membutuhkan waktu dan sumber daya yang besar. Applicant Tracking System (ATS) digunakan untuk mengotomatisasi proses tersebut, terutama dalam melakukan pencocokan kata kunci kompetensi dengan kebutuhan pekerjaan. Penelitian ini mengimplementasikan algoritma Aho-Corasick sebagai metode multiple string matching untuk melakukan screening kata kunci pada CV yang berformat PDF. Sistem dibangun menggunakan bahasa pemrograman C++ dan terdiri dari proses ekstraksi teks PDF, pembangunan automaton Aho-Corasick, serta pencarian kata kunci kompetensi secara otomatis. Pengujian dilakukan menggunakan dataset CV dari Kaggle dengan beberapa kelompok kata kunci yang mewakili bidang pekerjaan berbeda. Hasil pengujian menunjukkan bahwa pembangunan trie Aho-Corasick dapat dilakukan dengan waktu rata-rata sekitar 2 milisekon, sedangkan proses screening CV membutuhkan waktu rata-rata sekitar 33 milisekon per dokumen dengan total waktu pemrosesan sekitar 3 detik termasuk ekstraksi PDF. Hasil tersebut menunjukkan bahwa algoritma Aho-Corasick mampu melakukan pencarian banyak kata kunci secara efisien dan dapat diterapkan sebagai komponen screening otomatis pada Applicant Tracking System. (*Abstract*)**

Keywords—component; Applicant Tracking System, Aho-Corasick, string matching, curriculum vitae, keyword screening, automasi rekrutmen

I. Introduction (*Heading 1*)

Dalam era digital dan transformasi industri 4.0, proses rekrutmen perusahaan mengalami perubahan dalam proses. Seiring meningkatnya jumlah pelamar kerja yang dikirimkan secara online, proses pelamara memiliki tantangan untuk menyeleksi kandidat terbaik yang dengan jumlah waktu yang ketat. Proses screening manual terhadap puluhan atau bahkan ratusan CV menjadi beban administratif yang dikatakan signifikan, memakan waktu, dan rentan terhadap kesalahan manusia. Oleh karena itu, sistem otomasi dalam proses rekrutmen menjadi semakin penting dan relevan.

Applicant Tracking System (ATS) adalah solusi teknologi yang dirancang untuk mengelola dan mengotomatisasi proses rekrutmen. Salah satu fungsi kritis ATS adalah kemampuan untuk melakukan screening otomatis terhadap CV pelamar berdasarkan kriteria tertentu, khususnya identifikasi kehadiran kata kunci kompetensi yang relevan dengan lowongan pekerjaan. Namun, proses pencarian dan pencocokan kata kunci dalam jumlah besar pada dokumen teks masih menghadapi kendala dalam hal efisiensi dan akurasi. Selain itu, tantangan tambahan muncul dalam ekstraksi teks dari dokumen CV yang umumnya berformat PDF, yang memerlukan preprocessing sebelum proses matching dapat dilakukan.

Algoritma Aho-Corasick menawarkan solusi yang tepat untuk mengatasi permasalahan pencarian dengan kata kunci lebih dari satu. Sebagai algoritma yang dirancang khusus untuk pencarian multiple string matching secara simultan pada teks besar, Aho-Corasick memiliki kompleksitas waktu linear O(n+m+z), di mana n adalah panjang teks, m adalah panjang total pola yang dicari, dan z adalah jumlah kecocokan yang ditemukan. Algoritma ini dibangun dengan struktur data trie yang dikombinasikan dengan failure function dan dictionary link, memungkinkan pencarian all occurrences dari semua pattern dalam satu pass melalui teks. Algoritma ini jauh lebih efisien dibandingkan dengan pendekatan pencarian sekuensial tradisional, terutama ketika jumlah kata kunci kompetensi yang harus dicocokkan sangat banyak.

Makalah ini bertujuan untuk mengeksplorasi implementasi dari algoritma string matching pada screening cv menggunakan algoritma Aho-Corasick pada bagian *keyword matching* pada CV yang berbentuk PDF. Makalah ini berfokus pada menghitung waktu yang dibutuhkan untuk membuat Automaton aho-corasick dan melakukan screening berdasarkan Automaton Aho-Corasick.

II. Landasan Teori

A. Algoritma String Matching

String matching adalah suatu algoritma dalam ilmu komputer yang bertujuan untuk menemukan kehadiran satu atau lebih pola tertentu dalam sebuah string teks yang lebih besar. Secara formal, string matching didefinisikan sebagai masalah pencarian semua kemunculan dari pola P dalam teks T, di mana P dan T adalah string yang terdiri dari karakter-karakter dalam alphabet tertentu [1]. Dalam konteks praktis, string matching memiliki aplikasi yang sangat luas, mulai dari pencarian teks

dalam editor, analisis DNA dalam bioinformatika, deteksi plagiarisme, hingga sistem pencarian dalam *search engine*

Terdapat beberapa kategori string matching yang dapat diklasifikasikan berdasarkan jenis masalahnya. *Exact string matching* adalah kasus paling sederhana di mana pattern harus cocok secara identik dengan substring dalam teks. *Approximate string matching* memungkinkan adanya perbedaan kecil (misalnya satu atau dua karakter yang berbeda) antara pattern dan substring dalam teks. *Multiple pattern matching* adalah masalah yang lebih kompleks di mana terdapat lebih dari satu pattern yang harus dicari secara simultan dalam satu teks [2], [3].

B. Algoritma Aho-Corasick

Algoritma Aho-Corasick adalah algoritma string matching yang dirancang khusus untuk menyelesaikan masalah *multiple string matching*, yaitu pencarian serentak dari k pola dalam sebuah teks dalam satu kali perjalanan melalui teks [4]. Algoritma ini dipublikasikan oleh Alfred V. Aho dan Margaret J. Corasick pada tahun 1975 dalam paper berjudul "Efficient String Matching: An Aid to Bibliographic Search" dan telah menjadi salah satu algoritma pattern matching yang paling efisien dan banyak digunakan dalam berbagai aplikasi [4], [5].

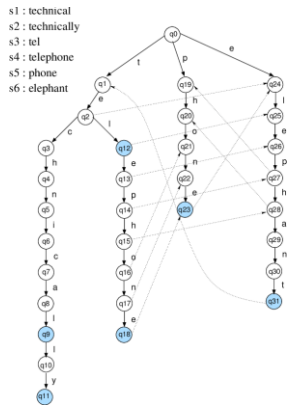
Algoritma Aho-Corasick dibangun atas tiga komponen utama

1. Trie (Prefix Trie): Sebuah struktut data pohon yang menyimpna semua pola secara compact, Trie bekerja dengan cara bebagi prefix yang sama antar pola. Setiap simpul pada trie mewakili satu karakter dan setiap jalur dari akar menuju daun merepresentasikan satu pola
2. Failure Function: Sebuah pointer yang menunujuk ke simpul trie lain yang yang merepresentasikan suffix terpanjang dari prefix saat ini yang juga merupakan prefix dari salah satu pattern. Failure function memungkinkan algoritma untuk melompat dari satu state ke state lain ketika terjadi mismatch, menghindari pemrosesan yang redundan
3. Output Link: Sebuah pointer yang menunujuk ke simpul terdekat yang merepresentasikan sebuah pola yang lengkap

Proses algoritma Aho-Corasick terdiri dari dua fase, fase preprocessing membuat trie dari setiap keywords dan fase matching untuk mencari dengan trie, berikut penjelesana masing-masing langkah

1. Fase preprocessing: Algoritma akan melakukan iterasi dari seluruh pola. Langkah pertama dalam algoritma Aho-Corasick adalah membangun struktur data Trie dari semua pattern yang akan dicari. Trie adalah pohon berakar di mana setiap edge diberi label dengan satu karakter, dan setiap path dari root ke node mewakili satu prefix dari pattern-pattern yang ada.

   Langkah selanjutnya melakukan BFS pada semua node untuk membangun Failure link. Failure link dilakukan secara rekursif. Sebuah simpul mencari failure link dari simpul leluhur nya. Melihat apakah dari parent terdapat simpul anak yang menyimpan karakter yang melakukan kesalahan ke mana. Pembangunan dimulai dari simpul setelah akar yang setiap simpul memiliki failure link yang mengarah ke akar.
2. Fase matching: Algoritma akan melakukan iterasi dari setiap karakter pada teks dan mencari dari akar trie untuk menjelajahi keberadaan pola di trie pada string pada teks. Setiap karakter dilakukan pengecekan apakah terdapat anak pada kondisi simpul terkini dan melakukan lompatan ke failure link jika tidak terdapat yang tepat dengan karakter.



Gambar 2.1 Contoh Trie untuk Aho-Corasick

Sumber: https://www.researchgate.net/figure/Building-an-Aho-Corasick-NFA-for-a-set-of-strings-Failure-transitions-to-only-non-q0_fig1_221046077

C. Curicculum Vitae

Curriculum vitae (CV) atau resume adalah dokumen formal yang berisi ringkasan informasi mengenai pendidikan, pengalaman kerja, keterampilan, dan prestasi seseorang. CV secara tradisional disimpan dalam format dokumen seperti Microsoft Word, PDF, atau format plain text, dan berfungsi sebagai instrumen utama yang digunakan oleh pelamar untuk mempresentasikan kualifikasi mereka kepada calon pemberi kerja

D. Applicant Tracking System (ATS)

Applicant Tracking System adalah solusi perangkat lunak terintegrasi yang dirancang untuk otomisasi, optimalisasi , dan membantu mengelola seluruh siklus proses rekrutmen dalam suatu organisasi. ATS mencakup berbagai fungsi yang dimulai dari publikasi lowongan kerja, penerimaan dan penyimpanan aplikasi pelamar, screening dan shortlisting kandidat, komunikasi dengan pelamar, hingga tracking dan pelaporan hasil rekrutmen [6], [7]. Dalam era transformasi digital, penggunaan ATS telah menjadi standar industri di perusahaan besar, karena memberi akses untuk menangani jumlah pelamar yang besar dengan efisien.

Salah satu fitur dalam ATS modern adalah screening berdasarkan kata kunci. Screening tersebut melakukan evaluasi otomatis dari CV pelamar dengan persyaratan pekerjaan berdasarkan ada atau tidaknya dan frekuensi kata dari kata kunci

pada CV. Screening berdasarkan kata kunci memungkinan ATS melakukan penyaringan diawal pada ribuan pelamar. Hal tersebut mengurangi beban pada screening manual dan mempercepat proses proses identifikasi kandidat yang berpotensi.

III. IMPLEMENTASI

Untuk implementasi, saya menggunakan Bahasa pemrograman c++. Program dibagi menjadi tiga buah file yaitu, main.cpp, Aho-Corasick.hpp, dan Aho-Corasick.cpp

File Aho-Corasick.hpp berisi deklarasi dari kelas AhoCorasick yang digunakan untuk pemroesan CV, Dengan pendefinisian dasar dari kelas dan method dari AhoCorasick. Juga terdapat deklarasi dari struct simpul trie Aho-Corasick yang bernama ACNode dan terdapat struktur kata kunci dan index ditemukan pada struct Match.

File AhoCorasick.cpp merupakan impelemntasi dari method yang berada di kelas AhoCorasick yang telah didefinisikan pada file AhoCorasick.hpp sebelumnya. File ini merupakan file utama untuk menagani pembanguanan trie. Terdapat 3 buah method utama yang diimplementasikan. Method insert(), merupakan method yang bertanggung jawab untuk membangun trie berdasarkan input kata kunci. Method build(), merupakan method yang bertanggung jawab dalam membangun failure link pada trie Aho-Corasick. Terakhir, Method search(), merupakan method yang berfungsi menelusuri string teks untuk mencari pola yang sesuai pada trie dan terdapat pada teks.

Terakhir, File main.cpp merupakan yang merupakan *entrypoin*t utama untuk kesuluruhan program. Pada main.cpp terdapat fungsi tambahan yang merupakan extractFromPDF(). Fungsi tersebut merupakan fungsi yang bekerja untuk mengambil informasi teks dari file cv yang berbentuk format PDF. Program akan dimulai dengan user memilih sebuah file txt berisi kata kunci. Kata kunci pada file tersebut akan digunakan untuk membangun sebuah trie Aho-Corasick. Program lalu akan membaca semua cv yang ada pada direktori cvs dan melihat berapa keyword yang cocok. File dibaca menggunakan fungsi extractFromPDF dan akan mengembalikan dalam bentuk string teks. String teks tersebut lalu akan dicari apakah terdapat pola kata kunci di dalam teks tersebut.

Dari Ketiga file tersebut, terdapat beberapa struct dan kelas yang merupakan komponen utama dari program ini.

A. *Struct ACNode*

```
struct ACNode{
    unordered_map<char, shared_ptr<ACNode>> children;
    shared_ptr<ACNode> fail = nullptr;
    shared_ptr<ACNode> dictlink = nullptr;
    string word = "";
};
```

Gambar 3.1 Definisi Struct ACNode

Struct ACNode merupakan represntasi dari simpul sebuah trie. Struct ini bertanggung jawab menyimpan informasi dari sebuah simpul. ACNode memiliki informasi dari simpul karakter apa yang terhubung setelah dirinya, simpul untuk melompat apabila tidak terdapat anak yang merupakan, dan string word jika dia merupakan simpul akhir sebuah pola.

B. *Kelas AhoCorasick*

Kelas AhoCorasick merupakan kelas utama yang berfungsi untuk memproses input pola dan menyimpan informasi dari simpul akar dari trie AhoCorasick. Kelas ini memilii tanggung jawab untuk membangun trie dan melakukan pencarian dari trie.

```
class AhoCorasick{
    private:
        shared_ptr<ACNode> root;
    public:
        AhoCorasick();
        void build();
        void insert(const string& word);
        vector<Match> search(const string& text);
};
```

Gambar 3.2 Definisi Kelas AhoCorasick

C. *Method insert()*

```
void AhoCorasick::insert(const string& word){
    auto node = root;

    for(char c : word){
        if(node->children.find(c) == node->children.end()){
            node->children[c] = make_shared<ACNode>();
        }
        node = node->children[c];
    }

    node->word = word;
}
```

Gambar 3.3 Method insert()

Method insert() merupakan method yang bertanggung-jawab untuk membangun trie pada kelas AhoCorasick. Fase Pembangunan pertama dari trie AhoCorasick dilakukan pada method insert(). Method insert() menerima sebuah string dan menelusuri trie yang sedang dibangun untuk melihat apakah terdapat huruf yang sesuai dari anak sebuah simpul. Bila terdapat sebuah huruf yang sesuai pada string, method akan menulusuri anak dari simpul tersebut dan mengiterasi ke huruf selanjutnya. Hal tersebut akan dilakukan terus menerus hingga bertemu huruf yang tidak merupakan anak simpul yang sekarang ditelusuri.

Method bila berada di sebuah simpul dan simpul tersebut tidak memiliki huruf tersebut pada anak simpulnya. Huruf tersebut akan dibuat sebagai simpul baru yang merupakan anak dari simpul yang sedang ditelusuri. Method lalu akan menulusuri simpul anak tersebut. Kedua kondisi tersebut dilakukan terus menerus hingga sampai ke huruf terakhir pada pola.

Ketika sudah sampai simpul huruf terakhir, method akan mengisi informasi kata apa yang dihasilkan setelah sampai simpul tersebut dengan string input yang dimasukkan.

D. *Method build()*

```
void AhoCorasick::build(){
    queue<shared_ptr<ACNode>> queue;

    for(auto const&[ch, child] : root->children){
        child->fail = root;
        queue.push(child);
    }

    while(queue.size() > 0){
        auto curr = queue.front();
        queue.pop();

        for(auto const&[ch, child] : curr->children){
            auto fails = curr->fail;

            while(fails && (fails->children.find(ch) == fails->children.end())){
                fails = fails->fail;
            }

            child->fail = fails ? fails->children[ch] : root;

            if (!child->fail->word.empty()) {
                    child->dictlink = child->fail;
                } else {
                    child->dictlink = child->fail->dictlink;
                }

            queue.push(child);
        }
    }
}
```

Gambar 3.4 Method build()

Method build() merupakan method yang bertanggung jawab dalam membangun failure link dan dictionary link. Setelah trie dibangun melalui metode insert(), method build() menggunakan pendekatan BFS untuk mengiterasi seluruh node dalam trie dan menghitung failure function serta dictionary link untuk setiap simpul

Proses konstruksi failure function dimulai dengan kedalaman pertama dari trie. Semua direct child dari node root diberi failure link yang menunjuk ke root, karena tidak ada suffix dari satu karakter yang dapat menjadi prefix dari pattern lain kecuali string kosong. Langkah berikutnya adalah iterasi BFS mulai dari level kedua dan seterusnya. Untuk setiap simpul anak dengan pointer meunju berlabel karakter 'c' dari simpul current , algoritma mencari simpul yang sesuai untuk di-set sebagai fail link. Hal ini dilakukan dengan mengikuti chain of fail links dari simpul current hingga ditemukan simpul yang memiliki anak dengan label karakter yang sama, atau sampai mencapai root. Jika simpul target ditemukan, fail link dari anak di-set ke anak dari simpul target tersebut. Jika tidak ada node yang cocok, fail link di-set ke root

Dictionary link, juga disebut output link atau dictionary suffix link, adalah optimization pointer yang digunakan untuk mengidentifikasi semua pola yang match pada posisi tertentu tanpa perlu melakukan traversal berulang kali. Untuk setiap simpu dalam trie, dictionary link di-set berdasarkan fail link-nya. Jika fail link dari suatu node menunjuk ke simpu yang merepresentasikan akhir dari sebuah simpul (node.word tidak kosong), maka dictionary link di-set langsung ke simpul fail tersebut. Jika simpul fail bukan merupakan akhir pattern, maka dictionary link di-set ke dictionary link dari simpul fail, menciptakan sebuah rantai yang menunjuk ke simpul pola terdekat yang dapat dicapai melalui failure links

E. *Fungsi search()*

```
vector<Match> AhoCorasick::search(const string& text){
    auto curr = root;
    vector<Match> result;

    for(int i = 0; i < text.size(); i++){
        char ch = text[i];
        while(curr != root && (curr->children.find(ch) == curr->children.end())){
            curr = curr->fail;
        }

        if (curr->children.find(ch) != curr->children.end()) {
            curr = curr->children[ch];
        } else {
            curr = root;
        }

        auto temp = curr;
        while(temp != nullptr){
            if(!temp->word.empty()){
                auto idx = i - temp->word.size() + 1;
                Match tempMatch;
                tempMatch.index = idx;
                tempMatch.word = temp->word;
                result.push_back(tempMatch);
            }
            temp = temp->dictlink;
        }
    }

    return result;
}
```

Gambar 3.5 Fungsi search()

Fungsi search() merupakan fungsi yang bertanggung jawab untuk melakukan pencarian pola pada string teks. Fungsi akan mengembalikan sebuah vector yang merupakan struct untuk menyimpan keywords dan index kemunculan pada string.

Fungsi akan melakukan looping pada semua karater pada string input. Fungsi akan memulai dari akar dan karakter pertama pada string input. Fungsi melihat anak dari simpul yang ditempati sekarang dan karakter yang sekarang dicocokan. Apabila cocok, akan lanjut menulusiri simpul dari anak yang cocok dan lanjut ke karakter selanjutnya. Jika tidak ada, akan menuju ke simpul failure links pada simpul yang ditempati sekarang, mengecek apakah ada anak yang sesuai di failure links dan akan lanjut ke karakter selanjutnya. Jika pada failure link gagal maka akan lanjut ke failure link milik simpul failure link selanjutnya. Hal tersebut akan dilakukan secara berulang hingga karakter terakhir.

Jika Fungsi menemukan simpul yang merupakan akhir dari suatu pola, maka ia akan menyimpan indeks karakter ditemukan pada string dan menyimpan kata yang ditemukan pada struct. Struct tersebut ditamabahakn pada vector result yang pada akhir akan dikembalikan sebagai hasil penemuan dari sebuah string input.

F. *Output program*

Program akan mengeluarkan output dari hasil pencocokan pola dengan waktu yang diperlukan untuk membangun trie dan waktu yang diperlukan untuk pencocokan dari sebuah string teks. Berikut merupakan contoh dari output program.

```
Available Keyword Entries:
1. embedde (embedde.txt)

Enter the number of the keyword profile you want to use: 1

Loading keywords from: embedde.txt
Successfully initialized automaton with 30 keywords.
>> Setup/Build Time: 2 ms
```

Gambar 3.6 Contoh output waktu Pembangunan Trie

```
Processed 120 CV(s) total.
>> Total Search & Extraction Time: 2669 ms (22 ms per CV average)
```

Gambar 3.7 Contoh output waktu pencocokan dan ekstraksi PDF

IV. PENGUJIAN

Pengujian dilakukan dengan dataset yang diambil dari https://www.kaggle.com/datasets/snehaanbhawal/resume-dataset?resource=download.

Dataset tersebut menyimpan ribuan CV dalam bentuk PDF dari berbagai jenis bidang keahlian. Uji Coba dilakukan dengan sebuah file txt yang berisi keywords dengan PDF yang berhubungan dengan keywords tersebut. Contoh misal sebuah keyword untuk pekerjaan embedded system akan diujicobakan pada folder yang berisi CV dengan pekerjan information system.

A. *Test Case 1*

Test case merupakan keywords yang berhubungan dengan low level software engineering seperti operating system dan network.

```
kernel
socket
pointer
interrupt
mutex
baremetal
bytecode
registers
packet
firmware
buffer
malloc
deadlock
syscall
endianness
assembly
volatile
daemon
thread
bootloader
compiler
driver
stack
heap
paging
payload
bitwise
protocol
ethernet
pipeline
```

Output program:

```
Available Keyword Entries:
1. embedde (embedde.txt)

Enter the number of the keyword profile you want to use: 1

Loading keywords from: embedde.txt
Successfully initialized automaton with 30 keywords.
>> Setup/Build Time: 2 ms
```

```
Processed 120 CV(s) total.
>> Total Search & Extraction Time: 2669 ms (22 ms per CV average)
```

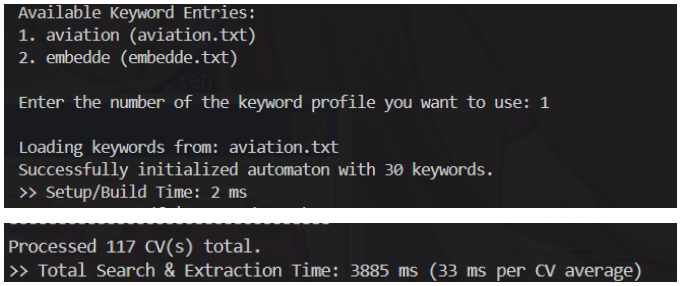
B. *Test Case 2*

Test case merupakan keywords yang berhubungan dengan penerbangan

```
pilot
avionics
boeing
airbus
faa
easa
atpl
aerodynamics
maintenance
mechanic
propulsion
navigation
air traffic
safety management
flight operations
cabin crew
dispatcher
hangar
logistics
cargo
turbofan
hydraulics
airworthiness
ifr
vfr
telemetry
meteorology
ground handling
```

```
    cockpit
ramp agent
```

Output program:

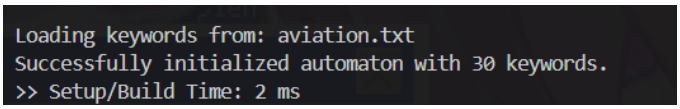


C. *Test Case 3*

Test case 3 merupakan keywrods yang berhubungan dnegan pekerjaan akutansi

```
    ledger
    audit
    gaap
    ifrs
    cpa
    taxation
    bookkeeping
    payroll
    reconciliation
    depreciation
    balance sheet
    cash flow
    revenue
    forecasting
    compliance
    invoice
    equity
    liability
    sarbanes-oxley
    quickbooks
    sap
    excel
    valuation
    capital
    amortization
    budgeting
    variance analysis
    accounts payable
    accounts receivable
financial reporting
```

Output program:



V. ANALISIS DAN KESIMPULAN

A. *Analisis*

Berdasarkan dari uji coba yang telah dilakukan pada bab sebelumnya. Program dapat menghasilkan sebuah trie dalam waktu rata-rata 2 milisekon. Program juga dapat mencari pada PDF dengan waktu rata-rata 3 detik. Masing-masing pengecekan pada masing-masing PDF memilki rata-rata waktu pencocokan 33 milisekon per PDF. Program juga menghabiskan waktu yang lebih banyak pada ketika CV memiliki jumlah kata yang sesuai lebih banyak.

B. *Kesimpulan*

Berdasarkan implementasi dan pengujian yang telah dilakukan, algoritma Aho-Corasick terbukti dapat digunakan secara efektif untuk melakukan screening kata kunci kompetensi pada Curriculum Vitae dalam Applicant Tracking System. Algoritma ini mampu melakukan pencarian banyak kata kunci secara simultan dengan efisien melalui struktur trie yang dilengkapi dengan failure link dan dictionary link.

Hasil pengujian menunjukkan bahwa proses pembangunan automaton Aho-Corasick memiliki waktu eksekusi yang sangat cepat, yaitu sekitar 2 milisekon. Sementara itu, proses pencocokan kata kunci pada dokumen CV memiliki rata-rata waktu sekitar 33 milisekon per dokumen, dengan total waktu pemrosesan sekitar 3 detik termasuk proses ekstraksi teks dari file PDF. Waktu pencarian juga dipengaruhi oleh jumlah kata kunci yang berhasil ditemukan pada dokumen, di mana semakin banyak kecocokan yang ditemukan maka semakin besar waktu pemrosesan yang diperlukan.

Dari hasil tersebut dapat disimpulkan bahwa algoritma Aho-Corasick merupakan solusi yang tepat untuk kebutuhan screening otomatis pada ATS karena mampu menangani pencarian multiple pattern secara cepat dan skalabel. Implementasi ini dapat membantu mengurangi beban proses seleksi manual serta mempercepat identifikasi kandidat yang memiliki kompetensi sesuai dengan kebutuhan perusahaan. Untuk pengembangan selanjutnya, sistem dapat ditingkatkan dengan menambahkan teknik normalisasi teks, pencocokan kata kunci berbasis sinonim, serta metode penilaian dan perankingan kandidat agar proses screening menjadi lebih akurat dan komprehensif

LAMPIRAN

https://github.com/testbored/Corasick-CV

UCAPAN TERIMA KASIH

Saya ingin mengucapkan terima kasih sebesar-besarnya terutama kepada Tuhan Yang Maha Esa, Allah Swt, yang telah menganugrahi saya kemampuan untuk bisa menyelesaikan makalah ini. Selanjutnya ucapan terima kasih juga ingin saya sampaikan kepada dosen saya Bapak Prof. Dr. Ir. Rinaldi, M.T. yang telah mengajarkan saya mata kuliah Strategi Algoritma. Terakhir, saya juga ingin mengucapkan terima kasih kepada orang tua dan teman-teman saya yang senantiasa memberikan saya dukungan dalam bentuk apa pun.

DAFTAR PUSTAKA

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

[2] D. E. Knuth, J. H. Morris, and V. R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 323–350, 1977.

[3] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, 1977.

[4] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[5] M. Crochemore and C. Hancart, "Pattern matching in strings," in *Handbook of Formal Languages*, vol. 2, G. Rozenberg and A. Salomaa, Eds. Berlin: Springer, 1997, pp. 539–633.

[6] G. Caruana, G. Cristianini, and A. Gini, "Online recruiting: The latest innovation in talent acquisition," *Business Intelligence Journal*, vol. 12, no. 2, pp. 123–135, 2019.

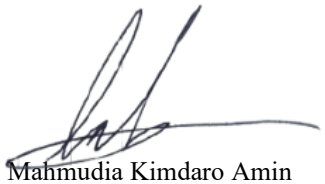
[7] M. Eckert and A. Chadha, "Product-based neural networks for user response prediction," in *Proceedings of the 3rd IEEE International Conference on Data Mining and Intelligent Information Technology Applications*, 2011, pp. 231–235

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Ttd



Mahmudia Kimdaro Amin

13524083