Pencarian Soal OSN-K Informatika pada File PDF dengan Label Tertentu Menggunakan Algoritma Aho-Corasick dan *Regular Expression*

Muhammad Nafis Habibi - NIM
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: author@gmail.com , author@std.stei.itb.ac.id

***Abstract*—Latihan soal adalah hal yang diperlukan untuk persiapan Olimpiade Sains Nasional (OSN). Makalah ini terfokus pada soal-soal OSN bidang informatika tingkat kabupaten. Soal-soal dengan materi atau label yang sama seringkali memiliki kesamaan kata atau pola pada kata. Maka dari itu, pencocokan string untuk mencari soal dengan label tertentu mungkin dapat dilakukan. Algoritma Aho-Corasick dan regular expression menjadi cara utama untuk pencocokan string pada implementasi.**

Keywords—string matching; aho–corasick; regex; osn informatika;

I. Pendahuluan

Olimpiade Sains Nasional (OSN) adalah olimpiade bergengsi yang diadakan oleh Pusat Prestasi Nasional Indonesia tiap tahunnya. Perlombaan dilakukan dalam beberapa tahap. Mulai dari tingkat kabupaten, provinsi, dan nasional. Berbagai bidang dilombakan dan salah satunya adalah informatika.

Soal OSN-K (tingkat kabupaten) informatika terdiri dari tiga bagian: abstraksi berpikir komputasional, pemecahan masalah komputasional, dan pemahaman algoritma dalam bahasa C++. Lebih sederhana lagi, soal OSN-K informatika juga bisa dibagi menjadi dua: soal analitika/logika dan soal algoritma. Soal analitika/logika biasanya dibuat dalam bentuk soal cerita.

Materi yang diujikan kurang lebih sama tiap tahunnya. Soal yang diberikan seringkali mirip dengan soal tahun-tahun sebelumnya. Beberapa soal dengan materi yang sama seringkali berisi kata-kata yang mirip juga. Jadi, mencari soal dengan materi tertentu mungkin bisa dilakukan dengan mengecek kata-kata yang muncul pada soal. Latihan soal dengan materi tertentu dapat lebih mudah jika soal-soalnya dapat dicari dengan mudah pula.

Pencocokan string bisa jadi solusi untuk mencari soal dengan materi tertentu. Pada makalah ini, pencocokan string dilakukan dengan algoritma Aho–Corasick dan *regular expression (regex)*. Arsip soal OSN yang dipublikasi oleh Tim Olimpiade Komputer Indonesia (TOKI) akan menjadi data uji untuk program yang akan dibuat.

Tujuan makalah ini adalah:

- Menerapkan pencocokan string dengan algoritma Aho Corasick dan *regex* untuk mencari soal dengan materi tertentu dalam file pdf dari soal OSN-K informatika.
- Membandingkan hasil pencarian dengan analisis mandiri.

II. Landasan Teori

A. Algoritma Aho-Corasick

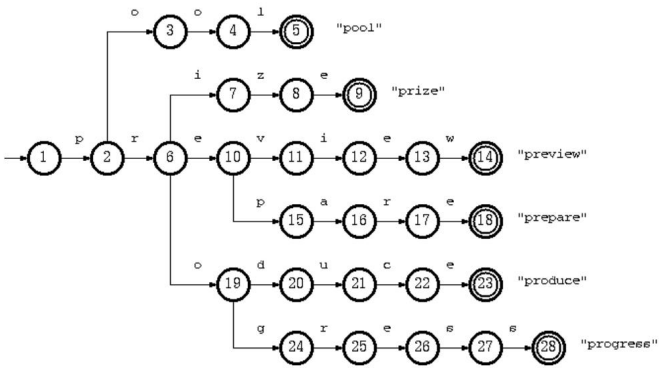
Algoritma Aho-Corasick merupakan salah satu algoritma pencocokan string yang cukup terkenal. Algoritma ini menggunakan struktur data trie yang dibangun menjadi sebuah automaton berupa suffix tree.

Trie adalah stuktur data berupa pohon yang memungkinkan pencarian informasi secara efisien. Kompleksitas waktu pencarian sebanding dengan panjang string input atau dengan kata lain, kompleksitas waktunya adalah O(M) dengan M adalah panjang dari string input. Namun, trie membutuhkan memori yang cukup besar untuk menyimpan informasi yang dibutuhkan [1]. Trie ini j

Trie juga termasuk *Deterministic Finite Automata* (DFA). Setiap simpulnya berkorespondensi dengan suatu state pada DFA [3], yaitu . Setiap simpul pada trie memiliki anak yang sisi menuju anaknya berkorespondendsi dengan karakter selanjutnya dari string *pattern* yang dimasukkan. Simpul yang merupakan simpul akhir dari string *pattern* perlu ditandai untuk menandakan adanya kecocokan saat pencarian. Simpul ini merupakan *final state* pada DFA.

Secara garis besar, struktur data trie adalah sebagai berikut:

```
class Trie {
    boolean output;
    Map<char, Trie> next;
}
```



Struktur data trie. Sumber: [3]

Menambahkan sebuah *pattern* ke dalam trie dapat dilaukan dengan mengiterasi tiap karakter pada *pattern*. Setiap karakter akan berkorespondensi dengan suatu pergerakan dari simpul *parent* ke simpul anaknya dimulai dari akar trie. Penelusuran dimulai dari karakter pertama dan akar trie. Jika simpul yang sedang ditelusuri tidak mempunyai anak yang berkorespondesi dengan karakter tersebut, simpul baru akan dibuat. Jika sudah ada, cukup telusuri simpul anak tersebut. Simpul terakhir yang ditelusuri akan ditandai sebagai simpul akhir, baik simpul daun (simpul tanpa anak) atau bukan (bisa terjadi jika *pattern* yang ditambahkan adalah *prefix* dari *pattern* sebelumnya yang pernah ditambahkan). Kedalaman dari trie akan bergantung pada panjang *pattern* terpanjang yang ditambahkan. Berikut pseudo code untuk menambahkan *pattern* pada trie.

Untuk pencarian apakah sebuah *pattern* ada pada trie, pencarian dapat dilakukan dengan cara yang mirip dengan penambahan *pattern*. String *pattern* diiterasikan bersamaan dengan trie. Untuk setiap karakter, cukup iterasi ke anak simpul yang berkorespondensi dengan karakter tersebut. Jika tidak ada simpul anak yang berkorespondensi atau simpul terakhir yang ditelusuri bukan simpul akhir, berarti *pattern* tersebut tidak ada di dalam trie. Jika simpul terakhir yang ditelusuri adalah simpul akhir, berarti *pattern* tersebut ada di dalam trie. Dengan pencarian seperti ini, trie bisa digunakan sebagai kamus untuk mengecek apakah sebuah kata ada berada di dalam kamus.

Trie dapat dikompresi menjadi lebih hemat memori dengan menggabungkan simpul yang hanya memiliki satu anak dengan anaknya. Namun, implementasi bisa lebih rumit.

Algoritma Aho-Corasick menggunakan struktur data trie yang dimodifikasi menjadi *suffix tree*. Pada *suffix tree*, setiap simpul juga menyimpan karakter yang menghubungkan *parent*-nya dengan dirinya serta tambahan sebuah *suffix link* (disebut juga *failure link*). *Suffix link* ini adalah sebuah sisi yang menuju simpul lain yang merupakan *longest proper suffix* dari string yang direpresentasikan oleh simpul tersebut. Terkhusus akar dari pohon, suffix link akan menunjukkan dirinya sendiri. *Suffix link* dari simpul anak dari akar pasti menunjukkan akar.

Secara garis besar, struktur data suffix tree adalah sebagai berikut:

```
class AhoCorasick {
    boolean output;
    Map<char, Trie> next;
    Trie parent;
    Trie suffixLink;
    char charFromParent;
}
```

Selain simpul-simpul tadi, *suffix link* dapat dicari dengan melihat *suffix link* dari *parent*-nya. Jika c adalah karakter yang berkorespondensi dengan sisi antara simpul v dengan *parent*-nya, suffix link dari simpul v bisa dengan melakukan transisi dari suffix link *parent*-nya ke simpul anak yang berkorespondensi dengan karakter c [2]. Pencarian suffix link dapat dilakukan secara rekursif ataupun dengan *Breath First Search* (BFS).

Pencarian kemunculan semua *pattern* yang muncul pada suatu teks dapat dilakukan mirip seperti trie biasa. Perbedaannya, saat mendapatkan kecocokan, seluruh simpul yang dapat ditelusuri melalui *suffix link* juga perlu dicek. Untuk mempercepat, *exit link* berupa sisi menuju simpul pertama yang juga berupa simpul akhir yang bisa didapat dengan menulusuri *suffix link* dapat dicatat.

Pseudocode untuk method pada algoritma Aho-Corasick adalah sebagai berikut:

```
function insert(Trie trie, string pattern) {
    Trie current = trie;
    for (char ch : pattern) {
        if (current.next[ch] == null) {
            current.next[ch] = new Trie();
            current.next[ch].parent = current;
            current.next[ch].charFromParent = ch;
        }
        current = current.next[ch];
    }
    current.output = true;
}
function next(Trie trie, char ch) {
    if (trie.next[ch] != null) {
        return trie.next[ch];
    } else {
        return next(trie.suffixLink(), ch);
    }
}
function link(Trie trie, char ch) {
    if (trie.suffixLink == null) {
        if (trie.parent == null) {
            trie.suffixLink = trie;
        } else if (trie.parent.parent == null) {
            trie.suffixLink = trie.parent;
        } else {
```

```
            trie.suffixLink =
next( link(trie.parent), trie.charFromParent);

        }

    }

    return trie.suffixLink;

}

function hasMatch(Trie trie, string input) {

    Trie current = trie;

    for (char ch : input) {

        current = next(current, ch);

        Trie temp = current;

        while (temp.parent != null) {


        }

    }

}
```

Kompleksitas waktu dari algoritma Aho-Corasick saat mencari kemunculan *pattern* pada suatu string input adalah O(M + K) dengan M adalah panjang string input dan Z adalah banyak kemunculan dari *pattern* yang ada.

B. Regular Expression

Regular expression (regex) adalah sebuah *finite automata* yang dapat menerima sebuah *regular language*. Regex dibuat dalam bentuk yang lebih "user-friendly" untuk mendeskripsikan sebuah *regular language* [4].

Language atau bahasa adalah himpunan string yang simbol atau karakternya berupa elemen dari himpunan alpabet. *Regular language* sendiri adalah *language* yang dapat didefinisikan dengan *finite automata*, seperti *regular expression*.

Regex memiliki beberapa operasi dasar, yaitu:

1) Union

Union atau gabungan dari dua bahasa L dan M adalah himpunan string yang merupakan elemen dari L atau M. Sama seperti gabungan pada teori himpunan. *Union* biasanya disimbolkan dengan L ∪ M.

2) Concatenation

Konkatenasi dari dua bahasa L dan M adalah himpunan string yang dapat dibuat dengan cara mengkonkatenasi string dari L dengan string dari M. Konkatenasi biasanya disimbolkan dengan titik/dot atau tanpa operator sama sekali.

3) Closure/Star

Klosur dari bahasa L adalah himpunan string yang dapat dibentuk dengan cara mengambil sejumlah string (boleh sama) dari L, lalu mengkonkatenasinya. Klosur disimbolkan dengan bintang, seperti L*.

Pengelompokkan dengan tanda kurung diperlukan untuk mengurutan operasi yang dilakukan. Dari tiga operasi dasar tersebut, urutan prioritas operasi yang dilakukan berturut-turut adalah *closure*, *concatenation*, lalu *union*.

Dengan ketiga operasi tersebut dan tanda kurung, *regular language* dapat didefinisikan secara induktif. Untuk setiap regex E, bahasa yang direpresentasikannya yaitu L(E).

Basis induksi adalah sebagai berikut:

- String kosong ϵ dan himpunan kosong $\emptyset$ adalah *regular expression* dengan bahasa $L(\epsilon) = \{\epsilon\}$ dan $L(\emptyset) = \{\emptyset\}$.
- Jika *a* adalah sebuah simbol apapun, maka *a* adalah *regular expression* dengan bahasa L(a) = {a}.
- Variabel dapat merepresentasikan bahasa apapun. Biasanya dituliskan dalam huruf besar, contohnya *L*.

Terdapat empat langkah induksi, satu untuk tiap operasi, dan satu lagi dari tanda kurung.

- Jika E dan F adalah *regular expression*, maka E + F adalah *regular expression* yang merepresentasikan gabungan bahasa dari E dan F. L(E + F) = L(E) ∪ L(F).
- Jika E dan F adalah *regular expression*, maka EF adalah *regular expression* yang merepresentasikan konkatenasi bahasa dari E dan F. L(EF) = L(E)L(F).
- Jika E adalah *regular expression*, maka E* adalah *regular expression* yang merepresentasikan klosur dari bahasa dari E. L(E*) = (L(E))*.
- Jika E adalah *regular expression*, maka (E) adalah *regular expression* yang sama dengan E. L((E)) = L(E).

Mesin regex yang disediakan oleh bahasa pemrograman biasanya memiliki fitur yang lebih banyak dibandingkan regex dasar ini. Beberapa fitur yang cukup penting yaitu:

- *Quantifier*. Dibandingkan hanya klosur untuk nol atau lebih pengulangan, terdapat beberapa *quantifier* tambahan seperti nol atau satu, satu atau lebih, tepat suatu angka, interval dari satu angka ke angka lain, dan suatu angka atau lebih.
- Kelas karakter. Bisa membuat himpunan karakter yang digunakan, misal huruf A s.d. F ditambah dengan huruf S. Bisa juga pakai himpunan yang telah dibuat, seperti alfanumerik, huruf, angka, spasi, dsb.
- Posisi karakter. Bisa dicek apakah karakter pertama atau akhir.

III. Implementasi

Implementasi dilakukan dalam bahasa pemrograman Java dengan JavaFX sebagai GUI-nya.

A. Algoritma Aho-Corasick

Seperti pada teori, algoritma Aho-Corasick diimplementasikan dengan struktur data trie tanpa dikompresi yang ditambahkan *suffix link* menjadi *suffix tree* dan ditambah dengan *exit link* untuk mempercepat pencocokan. Setiap simpul memiliki array sebesar 128 yang menunjukkan ke anak-

anaknya. Jadi, trie hanya dapat mengecek input karakter dalam kode ASCII standar. Jika suatu indeks pada array tidak ada simpul anaknya, maka nilainya dibuat menjadi null terlebih dahulu.

Trie yang berbeda dibuat untuk masing-masing label soal karena tiap label soal memiliki kata kuncinya masing-masing. Kata kunci dimasukkan ke dalam file json. Setiap label menyimpan kata kuncinya dalam bentuk array. Berikut sebagian isi jsonnya sebagai contoh:

```
{
  "kombinatorika": [
    "banyak cara",
    "berapakah banyak",
    "berapa banyak",
    "kombinasi"
  ],
  "peluang": [
    "peluang"
  ],
  "directed acyclic graph": [
    "prasyarat",
    "urutan tertentu",
    "terlebih dahulu"
  ],
}
```

Kata kunci yang tepat tidak dianalisis secara mendalam. Kata kunci mungkin keluar di soal yang tidak berhubungan dengan label. Untuk label yang lebih kompleks, bisa digunakan regex.

Terdapat kelas matcher tersendiri yang berisi trie yang telah dibangun untuk memudahkan pencocokan string dengan fungsi getMatches() dan hasMatch().

B. Regular Expression

Regex dibuat dengan mesin regex yang dimiliki bahasa Java, yaitu dari library java.util.regex. Jawa sendiri menggunakan regex yang mengikuti *Perl Compatible Regular Expression* (PCRE).

Sama seperti implementasi Aho-Corasick, setiap label memiliki string regexnya masing-masing dibandingkan hanya kata kuncinya. Setiap label bisa memiliki lebih dari satu string regex karena bisa jadi ada beberapa pola string yang menandakan label yang sama.

String regex juga disimpan di dalam file json. Berikut sebagian isi jsonnya sebagai contoh:

```
{
  "aritmetika modulo": [
    "\\d{1,}\\s+(mod|%)\\s+\\d{1,}"
  ],
  "rekursi": [
    "(\\w+\\().*?\\1"
  ],
  "fpb": [
    "(\\((\\w+),\\s+(\\w+)\\s?%\\s?(\\2)\\))|(\\((\\w+)\\s?%\\s?(\\w+),\\s+(\\7)\\))"
  ],
  "soal beranak": [
    "\\d{1,}\\s+((s\\.d\\.)|(sampai dengan)|(hingga)|(sampai))\\s+\\d{1,}\\.?$",
    "Soal \\d{1,}-\\d{1,}\\."
  ]
}
```

Pada implementasi regex, juga terdapat kelas matcher tersendiri yang berisi string-string regex untuk memudahkan pencocokan string dengan fungsi getMatches() dan hasMatch().

C. Pemrosesan File PDF

Bagian tersulit dalam implementasi adalah pemrosesan file PDF. Tidak seperti file teks biasa, file PDF tidak dapat dibaca begitu saja. Terdapat banyak elemen yang ditambahkan pada file PDF. Sebuah *library* Java digunakan untuk pemrosesan ini, yaitu Apache PDFBox.

PDF yang dibuka ditampilkan pada aplikasi sebagai gambar. Tiap halaman pada PDF dikonversi menjadi gambar, lalu ditambahkan pada scene JavaFX sebagai ImageView. Sayangnya, library PDFBox memberikan gambar dalam kelas BufferedImage yang dimiliki oleh *library* Java AWT yang bisa terbilang *outdated*. Bagusnya, ada kelas yang dapat membantu mengonversi BufferedImage ke WritableImage dari JavaFX. Pengonversian halaman PDF ke gambar menjadi cukup berat karena terdapat dua tahapan.

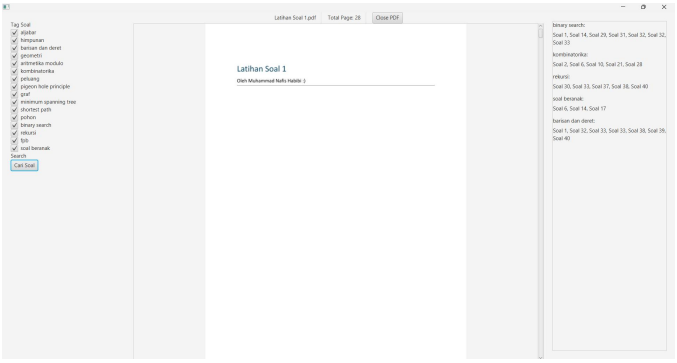
Pencocokan string dilakukan pada "raw text" yang didapatkan dari pemrosesan oleh kelas PDFTextScrapper. Sebelum melakukan pencocokan string, teks PDF dibagi terlebih dahulu per soalnya. Untuk membagi teks menjadi substring soal, dilakukan pencocokan string dengan regex untuk mencari awalan dari tiap soal.

D. Pencocokan String pada Soal

Setelah mendapatkan substring tiap soal, substring tersebut dimasukkan sebagai parameter dari matcher yang telah dibuat. Untuk setiap label, substring dimasukkan sebagai parameter dari matcher yang berkorespondensi dari tiap label. Jika terdapat match, berarti soal tersebut memiliki label tersebut. Hanya label-label yang dipilih sebelum pencocokan yang akan dicek dengan pencocokan string.

Matcher akan mengembalikan list label dari tiap soal dan list soal dari tiap label sehingga dapat dicari soal ini labelnya

apa saja dan label ini soalnya apa saja. Hasil ditampilkan sebagai string sederhana pada aplikasi.



Tampilan dari aplikasi. Di kiri ada pemilihan label. Di tengah ada tampilan PDF. Di kanan ada hasil pencocokannya.

IV. Hasil dan Pengujian

Pengujian dilakukan pada soal-soal OSN-K informatika yang diarsipkan di web resmi TOKI. Ditambah dengan 3 paket soal yang penulis buat sendiri. Terdapat beberapa faktor yang dicatat dalam pengujian ini:

- Jumlah pelabelan yang benar
- Jumlah pelabelan yang salah
- Jumlah soal yang tidak terdeteksi dengan label materi. Hanya label tipe soal (soal beranak, pilihan ganda, isian singkat, dan benar salah).

Sebagai pengetahuan tambahan, tipe soal OSN-K informatika telah berubah beberapa kali. Terakhir kali, soal 2024 mengubah format dari soal analitika pilihan ganda, lalu isian singkat, dan terakhir membaca program menjadi soal berpikir komputasional yang bentuknya campuran, dilanjutkan dengan soal permasalahan komputasional dan membaca program yang beranak tiga dan bentuknya campuran. Soal isian singkat baru muncul di tahun 2021. Sebelumnya hanya ada pilihan ganda. Sejak 2021 pula, program yang dibaca diubah dari bahasa Pascal menjadi bahasa C++. Perbedaan format sedikit mengubah pola soal-soal yang keluar pada OSN-K. Hal ini bisa meningkatkan kompleksitas program yang perlu dibuat.

Adapun hasil pengujian untuk tiap paket soal dapat dilihat pada tabel berikut:

No.	Paket Soal	Jumlah Soal	Jumlah Pelabelan Benar	Jumlah Pelabelan salah	Soal Tanpa Label Materi
1	OSN-K 2025	40	59	19	18

Adapun statistik dari tiap label dapat dilihat pada tabel berikut:

No.	Label	Pelabelan Benar	Pelabelan Salah	Persentase Benar
1.	Soal beranak	10	0	100%
2.	Pilihan ganda	7	1	88%
3.	Isian singkat	29	1	97%
4.	Benar salah	2	0	100%
5.	Himpunan	0	0	0%
6.	Barisan dan Deret	0	0	0%
7.	Aritmetika modulo	0	0	0%
8.	Kombinatorika	4	8	33%
9.	Peluang	0	0	0%
10.	Graf	4	1	80%
11.	Minimum Spanning Tree	0	0	0%
12	Directed acyclic graph	1	0	100%
13.	Shortest Path	0	0	0%
14.	Binary Search	0	0	0%
15.	Rekursi	2	7	22%
16.	FPB	1	0	100%
17.	Dynamic Programming	0	2	0%
	Total	60	20	75%

Beberapa halangan yang didapat saat melakukan pencocokan string:

- Ketika terdapat list 1, 2, 3, dst. selain pada nomor soal, seperti saat menjelaskan langkah atau menuliskan peraturan lomba, soal nomor 1, 2, 3, dst. menjadi mendapatkan label yang tidak seharusnya.
- Ketidaktahuan konteks dari soal membuat banyak kesalahan terjadi saat mengecek label kombinatorika dan rekursi. Menanyakan "berapa banyak" tidak mesti kombinatorika. Memanggil fungsi setelah didefinisikan juga tidak mesti rekursi, bisa jadi hanya pemanggilan fungsi pada fungsi lain.
- Soal dengan materi tertentu tidak mesti memiliki kata kunci yang diharapkan sehingga tidak terdeteksi sama sekali.

Hal yang bisa diambil dari hasil pengujian di atas:

- Pencocokan string cocok untuk mengecek tipe dari soal (beranak, pilihan ganda, isian singkat, dan benar salah).

V. Kesimpulan dan Saran

Pencocokan string biasa dapat dilakukan untuk menentukan label dari soal. Namun, hanya dengan pencocokan string tanpa mengetahui konteks dari soal, hasil jadi kurang akurat, terutama untuk label yang jarang ada kata-kata yang sama antar soal atau ada kata-kata yang terlalu sering muncul. *Artificial intelligence* yang dilatih khusus untuk pelabelan soal atau bahkan LLM biasa mungkin lebih baik dalam mementukan label soal. Analisis mandiri oleh orang yang berpengetahuan masih lebih baik dalam menentukan label soal.

Hasil pelabelan dari pencocokan string dapat menjadi dasar awal untuk menentukan label dari soal. Pencarian soal dengan label tertentu bisa lebih cepat dengan ini, walau bisa jadi banyak yang terlewat atau banyak *false positive*.

Link Repositori Github

https://github.com/NafisKreatif/Pencarian-Soal-OSK-Informatika-Tertentu-dalam-PDF

Ucapan Terima Kasih

The Puji syukur kepada Tuhan Yang Maha Esa yang telah memberikan kesempatan dan karunia-Nya sehingga penulis dapat menyelesaikan makalah ini**.** Terima kasih juga bagi orang tua yang telah membimbing hingga titik ini. Terima kasih kepada Pak Rila Mandala selaku dosen kelas K-02 pada mata kuliah IF2211 Strategi Algoritma. Penulis juga berterima kasih kepada teman-teman dan pihak lainnya yang terus mendukung penulis.

Referensi

[1] E. Owen, "Aho – Corasick Algorithm in Pattern Matching," Makalah Utama untuk IF2211 Strategi Algoritma, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Bandung, Indonesia, Mei 2015. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF221_Strategi_Algoritma_2015_032.pdf. [Diakses: 18 Juni, 2026].

[2] "Aho-Corasick algorithm," Algorithms for Competitive Programming, 18 Apr. 2025. Tersedia: https://cp-algorithms.com/string/aho_corasick.html. [Diakses: 18 Juni, 2026].

[3] T. Karoonboonyanan, "An Implementation of Double-Array Trie," *Linux Thai*, 21 November 2018. [Daring]. Tersedia: https://linux.thai.net/~thep/datrie/datrie.html. [Diakses: 19 Juni 2026]

[4] J. E. Hopcroft, R. Motwani, dan J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd Edition. Boston, MA, AS: Addison-Wesley, 2006.

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

Muhammad Nafis Habibi
13524018