Perbandingan Deteksi Kemiripan Teks Menggunakan Algoritma Boyer-Moore dan Rabin-Karp

Matthew Mahendra - 13521007
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13521007@std.stei.itb.ac.id

***Abstrak*—Plagiarisme merupakan tindakan kecurangan berupa pencurian sebagian atau keseluruhan dari suatu karya yang dibuat oleh seseorang. Tindakan yang merugikan ini sering terjadi dalam ranah akademik khususnya dalam penulisan makalah ataupun penugasan. Salah satu cara untuk mendeteksi tindakan kecurangan akademik ini adalah dengan menggunakan aplikasi pendeteksi kemiripan teks. Untuk itu, penulis melakukan perancangan dan implementasi dari dua algoritma pencocokan string yaitu algoritma Boyer-Moore dan Robin-Karp yang dapat digunakan untuk mencari kemiripan teks dari suatu tulisan. Dari kedua algoritma tersebut dilakukan perbandingan hasilnya. Didapatkan bahwa algoritma Boyer-Moore dan algoritma Rabin-Karp dapat digunakan, dengan algoritma Boyer-Moore untuk menentukan letak kemiripan dan Rabin-Karp untuk kemiripan secara keseluruhan.**

Kata Kunci—Boyer-Moore; Rabin-Karp; Plagiarisme; Pencocokan String

I. PENDAHULUAN

Tindakan kecurangan di dalam dunia pendidikan hadir dalam banyak bentuk. Salah satu bentuk yang paling sering terjadi adalah dalam bentuk plagiarisme. Plagiarisme biasanya terjadi dengan cara melakukan penyalinan sebagian maupun keseluruhan dari karya yang dibuat oleh orang lain. Tindakan plagiarisme ini biasanya dilakukan untuk mempermudah suatu tugas ataupun ujian.

Akan tetapi, tindakan tersebut merugikan institusi pendidikan yang bersangkutan. Hal ini disebabkan karena berkurangnya reputasi dari tulisan yang dikeluarkan oleh institusi tersebut serta integritasnya. Untuk itu, banyak institusi pendidikan akan menggunakan aplikasi untuk melakukan deteksi kemiripan teks dengan harapan sebelum tugas ataupun ujian yang mengandung kemiripan yang sangat besar dengan suatu teks yang tersedia di internet, maka akan terdeteksi dan ditindaklanjuti.

Untuk mewujudkan hal ini, penulis akan mencoba untuk membuat dan membandingkan dua algoritma pencocokan string yaitu algoritma Boyer-Moore dan Robin-Karp. Untuk membantu proses deteksi, maka akan diberikan juga algoritma Levenshtein Distance untuk menghitung rasio kemiripan dari dua teks.

II. DASAR TEORI

A. *Plagiarisme*

Menurut KBBI Daring, plagiarisme adalah penjiplakan yang melanggar hak cipta. Dari kata dasarnya, menurut KBBI Daring, plagiat adalah pengambilan karangan (pendapat dan sebagainya) orang lain dan menjadikannya seolah-olah karangan (pendapat dan sebagainya) sendiri, misalnya menerbitkan karya tulis orang lain atas nama dirinya sendiri. Dari definisi tersebut, plagiarisme dapat diartikan sebagai tindakan mengambil dan menyalin, baik sebagian maupun keseluruhan karya seseorang lalu menjadikannya sebagai karya sendiri.

Selain menyalin karya orang lain, plagiarisme juga dapat dilakukan pada diri sendiri. Tindakan ini disebut sebagai *self-plagiarism* [1]. Tindakan ini akan menggunakan ulang karya ilmiah yang telah dibuat sebelumnya, dengan sedikit perubahan, lalu dimasukkan ke dalam suatu karya ilmiah yang baru.

Dengan melakukan plagiarisme, maka seseorang telah melakukan tindakan pencurian terhadap karya yang telah dibuat oleh orang lain. Selain melakukan pencurian hasil dari plagiarisme akan memiliki kredibilitas yang harus dipertanyakan. Hal ini disebabkan karena penulis hanya menyalin dan bisa saja tidak memahami apa yang dituliskannya [2].

B. *Aplikasi Kemiripan Teks*

Untuk mengatakan suatu karya adalah hasil dari tindakan plagiarisme merupakan wewenang dari seorang yang berwenang, misalkan guru ataupun dosen. Oleh sebab itu, untuk membantu penilaian terhadap suatu tindakan plagiat, biasanya akan menggunakan aplikasi kemiripan teks.

Salah satu aplikasi yang terkenal di dunia pendidikan untuk melakukan pendeteksi kemiripan adalah aplikasi Turnitin. Aplikasi ini akan mengembalikan kemiripan dari masukan pengguna dengan mencocokan pada basis data yang terdiri dari publikasi-publikasi di Internet, sumber masukan pengguna yang lain, dan konten akademik lainnya [3]. Untuk setiap kemiripan yang ditemukan, akan diberikan penilaian seberapa mirip tulisan yang ada dengan basis data yang disediakan.

Hasil dari kemiripan tersebut akan memberikan informasi berupa teks yang memiliki kemiripan, sumber kemiripan, dan nilai kemiripan dalam persentasenya.

C. *Algoritma Boyer-Moore*

Algoritma Boyer-Moore merupakan algoritma pencarian string yang dikembangkan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977. Algoritma ini mencoba untuk memperbaiki algoritma Knuth-Morris-Pratt yang memiliki kompleksitas linear dan pergeseran yang kurang efektif. Selain itu untuk kasus subpattern yang lebih banyak, maka akan menghasilkan kompleksitas kuadratik.

Karena algoritma Knuth-Morris-Pratt memeriksa sebuah teks dari depan sehingga bisa saja informasi mismatch pola dan teks di akhir tidak dideteksi terlebih dahulu, maka untuk algoritma Boyer-Moore pemeriksaan dilakukan dari belakang pola. Dengan melakukan pemeriksaan dari belakang, diharapkan informasi tentang kesalahan dapat terdeteksi dengan lebih cepat. Pemeriksaan dari belakang ini disebut sebagai *looking-glass technique*.

Melalui pemeriksaan itu, maka ada tiga kondisi yang dapat terjadi. Kondisi tersebut adalah:

1. Pola dan teks ditemukan terus tanpa kesalahan
2. Ada perbedaan antara pola dan teks pada karakter yang ada pada pola
3. Ada perbedaan antara pola dan teks pada karakter yang tidak ada pada pola

Kondisi pertama dapat dicapai dengan cara melakukan iterasi dari belakang pola hingga awalan pola. Jika tidak ada kesalahan, maka dapat dikatakan bahwa pola sudah ditemukan pada teks. Pada kondisi kedua, maka diperlukan proses penggeseran teks secara cerdas berdasarkan lokasi kemunculan terakhir suatu karakter di pola yang ingin dicari pada teks. Pada kondisi ketiga, karena kasus tersebut secara implisit memberitahu kita bahwa tidak mungkin akan ada pola tersebut pada subbagian di teks, maka penggeseran teks dilakukan sejauh ukuran pola, agar mencarinya di bagian berikutnya. Teknik pemeriksaan ini disebut sebagai *character-jump technique*.

Karena untuk kasus kedua dan ketiga diperlukan informasi posisi terakhir dari suatu karakter dalam suatu pola, maka perlu dilakukan pembangkitan tabel yang mengandung informasi tersebut. Tabel ini disebut sebagai tabel *last occurrence*. Untuk karakter yang tidak terkandung dalam pola, maka nilai pada tabel *last occurrence* diberi nilai -1 yang akan berguna untuk proses *character-jump*. Pengambilan nilai dari tabel *last occurrence* dilakukan melalui fungsi *lo(x: char)* yang akan mengembalikan nilai karakter *x* dalam tabel *last occurrence*.

Pseudocode untuk algoritma ini dapat dilihat pada gambar 1.

D. *Algoritma Rabin-Karp*

Algoritma Rabin-Karp merupakan algoritma pencarian string yang dikembangkan oleh Richard M. Karp dan Michael O. Rabin pada tahun 1987. Algoritma ini menggunakan proses

```
function BMMatch(text, pattern: String) -> boolean
        Kamus Lokal
        n, m, i, j, lo : Integer
        LastOccurence : Array of Char

        Algoritma
        /* Length of Text */
        n <- length(text)
        /* Length of Pattern */
        m <- length(pattern)
        /* Index of Text */
        i <- m-1

        if(i > n-1) then
                -> false
        else
                /* Index of Pattern */
                j <- m-1
                do
                        if(pattern[j] = text[i]) then
                                if(j = 0) then
                                        -> true
                                else
                                        i <- i-1
                                        j <- i-1
                        else
                                lo<lastOccurence[text[i]]
                                i <- i + m - min(j, lo+1)
                                j <- m-1
                until (i <= n-1)
                -> false
```

Gambar 1. Pseudocode Algoritma Boyer-Moore

hashing dalam melakukan proses pencocokan untuk melakukan penyaringan teks manakah yang sama dengan pola dan yang tidak sama dengan pola. Algoritma ini merupakan algoritma yang dapat melakukan pencarian pola yang banyak dalam satu teks [5]. Oleh sebab itu, algoritma ini banyak digunakan dalam melakukan deteksi kemiripan dalam suatu teks yang cukup banyak misalnya antar dua dokumen.

Proses *hashing* yang digunakan dapat diatur sedemikian sehingga tingkat keakurasian dapat ditingkatkan dalam proses pencocokan pola dengan teks. Elemen yang dilakukan *hashing* adalah string dari pola dan juga dari teks yang akan diperiksa. Agar tidak terdapat nilai *hashing* yang sama, maka basis nilai *hash* adalah nilai prima yang cukup besar.

Untuk melakukan pendeteksian kemiripan yang baik, maka diperlukan proses penyaringan untuk kata-kata yang selalu muncul dalam suatu teks misalnya kata konjungsi seperti ‘dan’, ‘atau’, ‘jika’, ‘maka’, dan lain sebagainya. Hal ini untuk mengurangi jumlah hashing yang diperlukan.

Setelah melakukan penyaringan dan juga *hashing* terhadap pola dan teks, nilai kemiripan antar dua teks dapat didapatkan melalui persamaan berikut,

$$P = (SH \times 2/(J1 + J2)) \times 100\% \quad (1)$$

dengan P adalah persentase kemiripan, SH adalah jumlah hash yang sama, J1 dan J2 adalah jumlah hash yang dilakukan pada pola dan teks.

Pseudocode untuk algoritma ini dapat dilihat pada gambar 2.

```
function RKSimilarity(text, pattern: Array of String) -> double
        Kamus Lokal
        hashedText, hashedPattern: Set of Integer
        HS: Integer
        ret: Double

        Algoritma
        foreach (String s in pattern) {
                hashedPattern.add(hash(s))
        }

        foreach (String s in text){
                hashedText.add(hash(s))
        }

        HS <- 0

        foreach(Integer ph in hashedPattern){
                foreach(Integer th: hashedText){
                        if(ph = th) then
                                HS++
                        else
                                /* DO NOTHING */
                }
        }

        ret <- (2.0 * HS) / (hashedPattern.size() +
hashedText.size()) * 100.0

        -> ret
```

Gambar 2. Pseudocode Algoritma Rabin-Karp

E. Algoritma Levenshtein Distance

Berbeda dengan algoritma Rabin-Karp yang dapat melakukan proses perhitungan kemiripan karena adanya hashing, karena algoritma Boyer-Moore hanya melakukan pencocokan string berdasarkan satu pola saja, maka keluarannya adalah pola tersebut ditemukan di dalam teks atau tidak. Untuk itu, tidak dapat ditentukan kemiripannya.

Untuk membantu proses perhitungan kemiripan antar dua kata setelah pencocokan oleh algoritma Boyer-Moore, maka digunakan Levenshtein Distance untuk menghitung kemiripan antar dua string.

Algoritma ini dikembangkan oleh Vladimir Levenshtein pada tahun 1965. Algoritma ini pada dasarnya mengukur perbedaan antara satu kata dengan kata yang lain dan berapa kali terjadi suatu perubahan, baik itu delesi, insersi, maupun perubahan karakter pada suatu kata untuk menghasilkan karakter yang lain.

Pemrosesan algoritma ini biasanya dilakukan menggunakan sebuah matriks dan menggunakan perhitungan rekursif dengan basis dan rekurens sebagai berikut,

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Gambar 3. Fungsi Levenshtein Distance (diambil dari: https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0)

Dari perhitungan tersebut, maka dapat dibentuk matriks untuk menentukan nilai dari Levenshtein Distance. Persentase kemiripan antara dua string dengan menggunakan Levenshtein Distance kemudian didapatkan dengan perhiutngan sebagai berikut,

$$\text{Rasio} = 1 - (\text{LD} / \max(\text{len}(\text{string1}), \text{len}(\text{string2})) \quad (2)$$

dengan Rasio adalah persentase kemiripan, LD nilai Levenshtein Distance, dan string1, string2 teks yang akan diperhatikan kemiripannya.

Contoh pembangkitan algoritma tersebut akan diperlihatkan menggunakan contoh kata “CAT” dengan “CIP”. Dihasilkan matriks sebagai berikut,

#	0	1	2	3
0	#	C	A	T
1	C	0	1	2
2	I	1	1	2
3	P	2	2	2

Gambar 4. Matriks Perhitungan Algoritma Levenshtein

Perhatikan kolom 1,1 untuk karakter C dan C. Menggunakan perhitungan levenshtein ditentukan nilai minimum dari (1,1). Karena (1,1) nilai minimumnya bukan 0, maka dilakukan pemeriksaan terhadap nilai minimum dari perhitungan untuk lev(i,j-1) + 1, lev(i-1,j) + 1, dan lev(i,j-1) + 1 dengan penjumlahan satu dilakukan jika karakter tidak sama.

Untuk kolom 1,1, karena nilai minimum dari bagian rekurens adalah 0, maka kolom 1,1 diisi dengan 0. Selanjutnya pengisian dilakukan untuk kolom yang lain pada baris yang sama. Pengisian dilakukan hingga semua baris terisi.

Setelah semua baris terisi, maka didapatkan nilai Levenshteinnya adalah 2, yang berarti diperlukan 2 karakter yang diubah agar menjadi sama. Nilai tersebut dapat diambil pada bagian kanan bawah matriks atau pada kolom dengan indeks panjang kata pertama -1 dan panjang kata kedua-1.

Dengan demikian rasio untuk kemiripan CAT dan CIP adalah 1 – (2/3) = 0.33 atau 33%.

III. Implementasi Algoritma pada Persoalan

A. Lingkungan Implementasi

Implementasi dilakukan pada lingkungan bahasa pemrograman Java menggunakan *Java Development Kit* (JDK) 19. Untuk itu, pengujian dibuat menggunakan paradigma pemrograman objek. Dibuat kelas Boyer-Moore, Levenshtein, dan Rabin-Karp untuk pembuatan setiap algoritma pada persoalan tulisan ini.

Implementasi bertujuan untuk mendapatkan hasil dari setiap algoritma dan membandingkannya antar setiap algoritma. Selain itu, juga dapat menentukan kebenaran untuk beberapa kasus pengujian yang akan dilakukan.

B. Kelas BoyerMoore

Kelas BoyerMoore merupakan kelas yang digunakan untuk melakukan pencocokan string berdasarkan algoritma Boyer-Moore. Pada implementasi ini, nilai dari *last occurrence* dibatasi pada tabel ASCII 128.

Spesifikasi atribut untuk kelas ini adalah sebagai berikut,

Nama Atribut	Tipe Atribut	Akses	Deskripsi
lastOccurence	Array of Integer	private	Nilai dari fungsi Last Occurence
pattern	String		Pola masukan
text	String		Teks pemeriksaan

Tabel 1. Atribut Kelas BoyerMoore

Spesifikasi method untuk kelas ini adalah sebagai berikut,

Nama Method	Tipe Method	Akses	Deskripsi
BoyerMoore	Konstruktor	public	Konstruktor kelas. Melakukan pembangkitan lastOccurence
BMMatch()	Boolean	private	Method untuk melakukan pemeriksaan terhadap pattern dan string

Tabel 2. Method Kelas BoyerMoore

C. Kelas Levenshtein

Kelas Levenshtein merupakan kelas implementasi algoritma levenshtein. Kelas ini digunakan untuk melakukan perbandingan kemiripan untuk algoritma BoyerMoore.

Kelas ini memiliki method static sehingga tidak memiliki atribut. Spesifikasi untuk method-method tersebut adalah sebagai berikut,

Nama Method	Tipe Method	Akses	Deskripsi
costOfSubstitution	Int	public	Menghitung nilai perubahan yang diperlukan dari dua masukan string
min	Int	public	Menghitung nilai minimum dari sekumpulan integer.
calculate	Double	Public	Menerima dua inputan string dan melakukan proses perhitungan algoritma Levenshtein. Mengembalikan nilai rasio

Tabel 3. Method Kelas Levenshtein

D. Kelas RabinKarp

Kelas RabinKarp merupakan kelas yang digunakan untuk melakukan pencocokan string berdasarkan algoritma Rabin-Karp. Fungsi hashing untuk algoritma ini menggunakan *built-in hash* yang dimiliki oleh kelas java.util.Objects dengan method hash().

Spesifikasi atribut untuk kelas ini adalah sebagai berikut,

Nama Atribut	Tipe Atribut	Akses	Deskripsi
text	List of String	private	Teks yang telah dipisahkan dan disimpan sebagai list
pattern	List of String		Teks yang telah dipisahkan dan disimpan sebagai list

Tabel 4. Atribut Kelas RabinKarp

Spesifikasi method untuk kelas ini adalah sebagai berikut,

Nama Method	Tipe Method	Akses	Deskripsi
RabinKarp	Konstruktor	public	Konstruktor kelas.
hash	Integer	private	Method untuk melakukan hashing terhadap pattern dan string. Menggunakan Objects.hash()
RKSimilarity	Double	Public	Method untuk melakukan perbandingan dua string berdasarkan algoritma Rabin-Karp.

Tabel 5. Method Kelas RabinKarp

E. Alur Program

Untuk kedua algoritma program akan menerima dua masukan string. Untuk kedua string, dibuat menjadi *uppercase* sehingga pencocokan tidak menjadi *case sensitive*. Untuk membantu proses pemeriksaan, maka akan dilakukan pemisahan menggunakan Regular Expression pada setiap titik dengan mengabaikan spasi pada implementasi algoritma Boyer-Moore dan memisahkan setiap kata untuk algoritma Rabin-Karp dengan mencoba untuk menghilangkan kata-kata seperti kata konjungsi yang selalu ada pada setiap dokumen.

Setelah dilakukan pemisahan menggunakan Regular Expression, maka selanjutnya dilakukan pencocokan dengan algoritma Boyer-Moore dan Rabin-Karp. Hasil yang diharapkan adalah melakukan perbandingan dan pada akhirnya menampilkan nilai kemiripan antara dua teks atau dokumen yang menjadi masukan.

Regular Expression yang digunakan untuk memisahkan teks dan pattern adalah sebagai berikut,

1. (?:\s)*\.(?:\s), untuk algoritma Boyer-Moore
2. \s, untuk algoritma Rabin-Karp

Perhitungan kemiripan untuk Boyer-Moore menggunakan perhitungan total ratio levenshtein / jumlah kesamaan, sedangkan untuk algoritma Rabin-Karp menggunakan rumus (1).

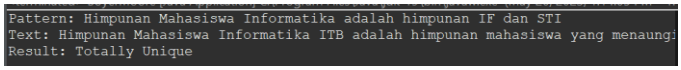
IV. Hasil Penelitian dan Pembahasan

Hasil Penelitian dilakukan dengan melakukan pengujian terhadap beberapa teks dan pattern yang sama untuk algoritma Boyer-Moore dan Rabin-Karp.

A. Pengujian Algoritma Boyer-Moore

Pengujian pertama menggunakan teks: “Himpunan Mahasiswa Informatika ITB adalah himpunan mahasiswa yang menaungi mahasiswa jurusan IF dan STI. Himpunan ini terletak di Labtek V ITB Ganesha.” dan pola: “Himpunan Mahasiswa Informatika adalah himpunan IF dan STI.”

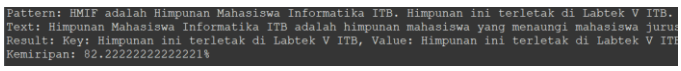
Dengan pola dan teks tersebut didapatkan hasil sebagai berikut,



Gambar 5. Pengujian 1 Boyer-Moore

Hasil menunjukkan untuk pengujian pertama bahwa pola dan pattern adalah beda sehingga result yang dikeluarkan adalah “*Totally Unique*” karena tidak ditemukan kesamaan sama sekali.

Untuk pola “Himpunan Mahasiswa Informatika adalah himpunan IF dan STI.”, didapatkan hasil sebagai berikut,



Gambar 6. Pengujian 2 Boyer-Moore

Hasil menunjukkan untuk pengujian kedua bahwa pola dan pattern memiliki kesamaan. Kesamaan terlihat pada teks Himpunan ini terletak di Labtek V ITB Ganesha dengan kemiripan 82%. Total kemiripan untuk kedua pola dan teks adalah sebesar 82.22% karena hanya ada 1 bagian saja yang sama.

B. Pembahasan Hasil Algoritma Boyer-Moore

Algoritma Boyer-Moore menjadi algoritma *exact string-matching* yang artinya hanya akan mendeteksi sebuah kesamaan pola pada teks jika dan hanya jika ada kesamaan yang tepat.

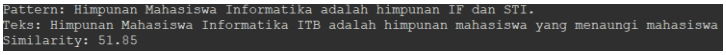
Pengujian kedua menghasilkan suatu kesamaan dikarenakan adanya pola yang tepat sama penulisannya seperti pada teks. Pada pengujian pertama meskipun secara isi dari pola memiliki arti yang sama, bentuk penulisannya tidak sama. Oleh sebab itu, algoritma Boyer-Moore tidak mendeteksi kesamaan tersebut.

Meskipun dapat mendeteksi, hasil kemiripan tidak terlalu akurat karena hanya menggunakan pembagian hasil Levenshtein dengan

C. Pengujian Algoritma Rabin-Karp

Pengujian pertama menggunakan teks dan pola yang sama seperti pada algoritma Boyer-Moore yaitu teks: “Himpunan Mahasiswa Informatika ITB adalah himpunan mahasiswa yang menaungi mahasiswa jurusan IF dan STI. Himpunan ini terletak di Labtek V ITB Ganesha.” dan pola: “Himpunan Mahasiswa Informatika adalah himpunan IF dan STI.”

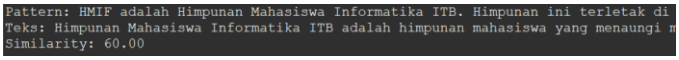
Dengan pola dan teks tersebut didapatkan hasil sebagai berikut,



Gambar 7. Pengujian 1 Rabin-Karp

Dari teks dan pola tersebut, maka didapatkan kemiripan total yaitu pada angka 51.85%. Algoritma ini langsung memeriksa keseluruhan teks dan tidak mengeluarkan kalimat mana yang membuatnya menjadi 51.85%.

Untuk pola kedua yaitu “Himpunan Mahasiswa Informatika adalah himpunan IF dan STI.”, didapatkan hasil sebagai berikut,



Gambar 8. Pengujian 2 Rabin-Karp

Dari teks dan pola tersebut, maka didapatkan kemiripan 60%.

D. Pembahasan Hasil Algoritma Rabin-Karp

Algoritma Rabin-Karp tidak memungkinkan untuk menemukan secara tepat letak kemiripan antara pola dan teks, akan tetapi langsung memberikan persentase kemiripannya. Dari pengujian yang pertama, didapatkan kemiripan 51.85%. Ini dikarenakan adanya hasil hashing yang terdapat dalam teks

seperti kata Himpunan, Mahasiswa, Informatika, IF, STI yang membuat nya memberikan hasil yang besar.

Untuk pengujian kedua, karena pada kata kedua adalah *exact match*, maka nilai persentasenya akan semakin besar yang memberikan nilai 60%.

V. Simpulan dan Saran

Dari hasil implementasi untuk kedua algoritma ini didapatkan kesimpulan sebagai berikut,

1. Algoritma Boyer-Moore bersama dengan Levensthein serta Algoritma Rabin-Karp dapat digunakan untuk melakukan pendeteksian kemiripan teks
2. Algoritma Boyer-Moore akan menjadi lebih tepat digunakan pada aplikasi pendeteksi kemiripan teks untuk menemukan posisi kemiripan yang terdeteksi karena sifatnya yang *exact matching*. Algoritma ini juga lebih tepat digunakan pada aplikasi pencarian riwayat berdasarkan masukan teks pengguna
3. Algoritma Rabin-Karp lebih tepat digunakan untuk pengecekan banyak pola pada suatu teks
4. Algoritma Levenshtein dapat digunakan sebagai pembantu penentuan kemiripan teks yang didasarkan pada jarak yang dibutuhkan untuk membuat kedua teks menjadi sama

Dari hasil implementasi ini juga diberikan saran untuk pengembangan ataupun penelitian yang serupa sebagai berikut,

1. Dapat mengembangkan algoritma Boyer-Moore untuk pendeteksian kemiripan khususnya pada bagian perhitungan kemiripan total
2. Dapat menggunakan fungsi hashing buatan sendiri untuk algoritma Rabin-Karp.

Tautan Video Penjelasan

Video dapat dilihat pada tautan berikut: https://youtu.be/XRkcSBArhTA

Ucapan Terima Kasih

Puji dan syukur diucapkan kepada Tuhan yang Maha Esa sehingga makalah ini dapat diselesaikan. Tidak lupa kepada Orang tua penulis yang telah mendukung proses pendidikan penulis sehingga makalah ini dapat disusun, kepada Ir. Rila Mandala, M. Eng., Ph.D. sebagai dosen pengampu IF2211 yang telah menurunkan ilmunya tentang Matematika Diskrit dan kepada seluruh kolega yang telah menyemangati proses pembuatan makalah ini.

References

[1] M. Ben, "Self-Plagiarism: How to Define It and Why You Should Avoid It," 8 Februari 2016. [Online]. Available: https://www.aje.com/arc/self-plagiarism-how-to-define-it-and-why-to-avoid-it/. [Accessed 17 Mei 2023].

[2] University of New South Wales Sydney, "What is Plagiarism?," 17 Juni 2022. [Online]. Available: https://www.student.unsw.edu.au/what-plagiarism. [Accessed 17 Mei 2023].

[3] Turnitin, "Does Turnitin detect plagiarism?," 5 Oktober 2022. [Online]. Available: https://www.turnitin.com/blog/does-turnitin-detect-plagiarism. [Accessed 17 Mei 2023].

[4] R. S. Boyer and J. S. Moore, "A Fast String Searching Algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762-772, 1977.

[5] R. E. Putri and A. P. U. Siahaan, "Examination of Document Similarity Using Rabin-Karp Algorithm," *International Journal of Recent Trends in Engineering & Research*, vol. 3, no. 8, pp. 196-201, 2017.

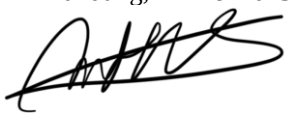
[6] E. Nam, "Understanding the Levenshtein Distance Equation for Beginners," Medium, 27 Februari 2019. [Online]. Available: https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0. [Accessed 20 Mei 2023].

[7] Program Studi Teknik Informatika STEI-ITB, "Homepage Rinaldi Munir," 2021. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf. [Accessed 20 Mei 2023].

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, 22 Mei 2023



Matthew Mahendra
13521007