Penerapan Algoritma Rabin-Karp dalam Mendeteksi Plagiarisme Source Code berdasarkan String Matching

Mgs. Muhammad Riandi Ramadhan - 13517080
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Indonesia
13517080@std.stei.itb.ac.id

***Abstrak*—Plagiarisme adalah salah satu masalah umum yang dapat ditemui pada suatu pembuatan karya. Plagiarisme adalah penjiplakan karangan, pendapat, dan sebagainya dari orang lain dan menjadikannya seolah karangan dan pendapat sendiri. *Source code* merupakan salah satu karya yang dapat memiliki hak cipta. Beberapa perangkat lunak mampu menilai tingkat plagiarisme dari suatu karya baik dokumen teks atau gambar. Algoritma Rabin-Karp mampu mendeteksi plagiarisme pada *source code* dengan metode *string matching*. Algoritma ini menggunakan fungsi hash sebagai dasar pembanding.**

Kata kunci—hash, string-matching, pola, teks.

A. Pendahuluan

Plagiarisme atau biasa disebut sebagai plagiat menurut Kamus Besar Bahasa Indonesia adalah penjiplakan atau pengambilan karangan, pendapat, dan sebagainya dari orang lain dan menjadikannya seolah karangan dan pendapat sendiri (KBBI, 1997:775). Sedangkan menurut Peraturan Menteri Pendidikan Nasional Republik Indonesia No. 17 Tahun 2010, plagiat adalah perbuatan secara sengaja maupun tidak sengaja dalam memperoleh atau mencoba memperoleh kredit atau nilai untuk suatu karya ilmiah dengan mengutip sebagian atau seluruh karya dan/atau karya ilmiah pihak lain yang diakui sebagai karya ilmiahnya tanpa menyatakan sumber secara tepat dan memadai.

Dengan kemajuan teknologi informasi, plagiarisme sangat mudah dilakukan oleh setiap pihak baik yang berkepentingan atau tidak. Internet merupakan sarana utama yang biasa digunakan pelaku plagiarisme untuk melakukan tindakan tersebut. Kemudahan yang diberikan dapat menanamkan rasa ‘candu’ untuk terus mengulangi hal yang sama. Kita masih menemui beberapa kasus terkait plagiarisme yang melibat berbagai pihak termasuk para civitas akademik. Plagiarisme dinilai mampu menurunkan tingkat kreativitas seseorang untuk berkarya. Oleh karena itu, beberapa pihak telah melarang dan mengatur hal ini agar tidak dilakukan dengan menerapkan peraturan tertulis sebagai ketentuan pembuatan suatu tugas atau karya.

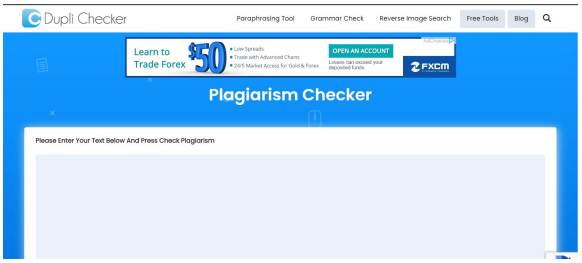
Plagiarisme tidak hanya dapat ditemukan dalam dokumen teks dan gambar tetapi juga terdapat pada *source code* program. Saat ini kita dapat dengan mudah mencari *source code* program di Internet pada berbagai website seperti Github dan sejenisnya. Kode tersebut biasanya dimanfaatkan untuk menjalankan suatu fungsi tertentu tergantung kebutuhan. Internet telah menyediakan *source code* secara lengkap dalam hampir semua bahasa pemrograman. Hal ini dapat kita temukan dengan mudah saat menjelajahi *browser* sehingga sesuatu yang berbentuk digital seperti *source code* dapat dengan mudah untuk disalin.

```
FUNCTION PBMAIN
    DIM ma4(1 TO 4,1 TO 4) AS DOUBLE, Det AS DOUBLE, T$, I&, J&
    ' matrix
    ma4(1,1) =  1 : ma4(1,2) =  3 : ma4(1,3) = -3 : ma4(1,4) =  5
    ma4(2,1) =  4 : ma4(2,2) =  2 : ma4(2,3) =  1 : ma4(2,4) =  2
    ma4(3,1) =  3 : ma4(3,2) =  2 : ma4(3,3) = -2 : ma4(3,4) =  2
    ma4(4,1) =  0 : ma4(4,2) =  1 : ma4(4,3) =  2 : ma4(4,4) = -1
    Det = 1
    CALL MakeResultsString(ma4(),4,Det,T$,"Original")
    CALL MatrixInversion(ma4(), 4 , Det)
    CALL MakeResultsString(ma4(),4,Det,T$,"Inverted")
    CALL MatrixInversion(ma4(), 4 , Det)
    CALL MakeResultsString(ma4(),4,Det,T$,"Inversion of inverted matrix = Original")
    MSGBOX T$,,"Results:"
END FUNCTION
'
SUB MatrixInversion(A() AS DOUBLE, M AS LONG, Determinant AS DOUBLE)
    ' Gauss reduction inversion method.
    ' M is the order of the square matrix A()
    ' A() inverse is returned in A().
    ' Determinant is returned.
    LOCAL I, J, K, L AS LONG, T AS DOUBLE, Pivot AS DOUBLE
    Determinant = 1
    FOR J = 1 TO M
        Pivot = A(J,J) : A(J,J) = 1
        Determinant = Determinant * Pivot
        IF Determinant = 0 THEN MSGBOX "Matrix singular " _
          + "- cannot invert",,"Problem": EXIT SUB
        '
        ' Divide pivot row with pivot element.
        FOR K = 1 TO M : A(J,K) = A(J,K) / Pivot : NEXT
        FOR K = 1 TO M
            ' Reduce the non pivot rows.
            IF K <> J THEN
               T = A(K,J) : A(K,J) = 0
               FOR L = 1 TO M : A(K,L) = A(K,L) - A(J,L) * T : NEXT
            END IF
        NEXT
    NEXT
END SUB
```

Gambar 1. Contoh *source code* dari suatu program
(sumber : https://ecstep.com)

Di samping itu, perkuliahan Teknik Informatika tidak dapat terlepas dari tugas pemrograman yang tentunya mengharuskan pembuatan program sesuai dengan spesifikasi. Plagiarisme dapat terjadi pada saat pelaksanaan tugas baik menyalin dari internet ataupun dari sesama teman. Hal ini tentunya bertentangan dengan ketentuan tugas dan etika perkuliahan. Dalam kasus buruknya, seseorang mengambil tugas orang lain dan mencantumkan namanya pada tugas tersebut. *Source code* yang disalin dapat berupa secara

keseluruhan dan sebagian. Untuk mendeteksi plagiarisme ini, banyak pemeriksa masih menggunakan metode secara manual dengan membandingkan dua atau lebih kode yang dilihat mirip. Tentunya ini akan memakan banyak waktu dan memungkinkan terjadinya kesalahan.



Gambar 2. www.duplichecker.com
(sumber : dokumen pribadi)

Saat ini kita dapat menemukan berbagai perangkat lunak baik berbasis web, desktop, atau pun mobile yang dapat digunakan untuk mendeteksi tingkat kemiripan untuk menghindari terjadinya plagiarisme. Kita dapat mendeteksi kehadiran bagian dari sebuah *source code* pada *source code* lainnya dengan menggunakan metode *string matching*. Sebagai salah satu algoritma *string matching*, algoritma Rabin-Karp yang memanfaatkan fungsi hash akan digunakan sebagai dasar utama untuk mendeteksi terjadinya plagiarisme pada *source code* program.

B. Landasan Teori

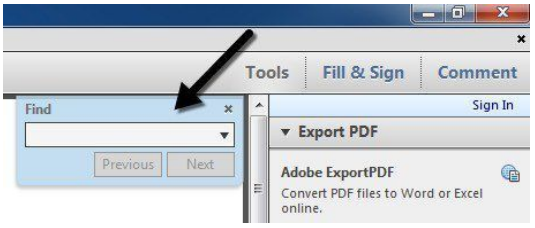
A. *String Matching*

Berdasarkan oleh *World Agreement Agenda* O812, *string* merupakan deretan simbol yang tidak tertentu panjangnya, yang dianggap sebagai panjang satu unit. *String* dapat tersusun atas beberapa hal, baik itu berupa huruf, angka, karakter khusus, maupun karakter *unicode*. *String matching* atau bisa disebut pencocokan *string* merupakan langkah pencarian kemunculan *string* pendek dengan ukuran n yang disebut sebagai *pattern* dalam suatu *string* panjang dengan ukuran m yang dinamakan *text*, dengan n bernilai lebih kecil atau sama dengan m.

Dalam pencarian *string* tersebut, kita dapat menggunakan berbagai jenis algoritma yang dapat disesuaikan berdasarkan pendekatan tertentu. Berikut ini adalah beberapa algoritma yang biasa digunakan pada *string matching*.

- Algoritma *brute force*
- Algoritma Boyer-Moore
- Algoritma Knuth-Morris-Pratt (KMP)
- Algoritma Aho-Corasick
- Algoritma Rabin-Karp

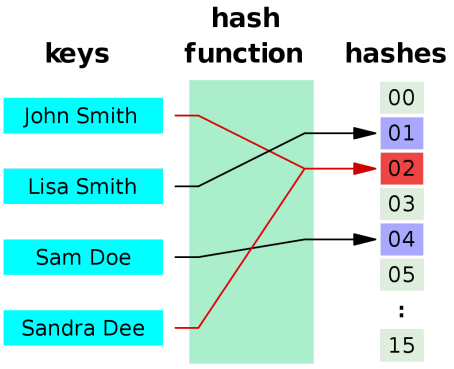
Berbagai perangkat lunak telah menggunakan *string matching* untuk melakukan pencarian. Dengan ini, kita dapat melakukan pencarian terhadap suatu karakter, kata, ataupun kalimat. Tidak hanya pencarian, pencocokan *string* juga dapat dilakukan untuk mengoreksi teks atau dokumen yang telah dibuat.



Gambar 3. Pencarian di Adobe Reader PDF
(sumber :www.online-tech-tips.com)

B. *Fungsi Hash*

Fungsi hash merupakan sebuah fungsi yang menerima masukan *string* yang panjangnya sembarang. Fungsi hash mentransformasikan *string* tersebut menjadi *string* yang berupa *output* dengan panjang tetap namun pada umumnya memiliki ukuran yang lebih kecil.



Gambar 4. Fungsi Hash
(sumber)www.online-tech-tips.com)

Berikut ini adalah persamaan dari fungsi hash.

$$h = H(M)$$

M : pesan ukuran sembarang

h : nilai hash

h <<<< M

Fungsi hash diimplementasikan untuk beberapa keperluan seperti mempercepat pencarian dalam tabel data atau perbandingan data di basis data, pencarian duplikasi atau kesamaan dalam sebuah arsip komputer yang besar, dan enkripsi pesan pada suatu aplikasi.

C. *Algoritma Rabin-Karp*

Algoritma Rabin-Karp merupakan sebuah algoritma pencarian *string* yang diciptakan oleh Richard M. Karp dan Michael O. Rabin yang menggunakan fungsi hash untuk menemukan kemunculan sebuah pola pada teks. Dalam algoritma Rabin-Karp. kita akan mendapatkan hasil dari fungsi hash dari pola yang kita cari dan mengecek jika hash

bergulir pada teks yang diperiksa cocok dengan pola yang ada atau tidak. Jika tidak cocok, maka kita bisa menyatakan bahwa pola tidak ditemukan atau terdapat di dalam teks yang diperiksa. Bagaimanapun juga, jika cocok, maka pola dipastikan berada di dalam teks.

Pada pencocokan string algoritma ini memiliki kompleksitas O(n+m) untuk mencari substring dengan panjang m dalam string dengan panjang n, namun algoritma ini memiliki kasus terburuk dengan kompleksitas O(nm). Algoritma ini menggunakan fungsi hash sebagai pembanding antara pola string yang dicari dengan substring pada teks, sehingga kecepatan eksekusi algoritma ini juga ditentukan oleh fungsi hash yang digunakan.

Algoritma Rabin-Karp memiliki kompleksitas sebesar O(m+n) untuk mencari pola dengan panjang n dalam teks dengan panjang m. Dalam kasus terburuk, pencari pola pada teks tersebut akan memiliki kompleksitas sebesar O(mn). Sebagaimana yang telah disebutkan sebelumnya, algoritma ini menggunakan fungsi hash untuk membandingkan pola dan teks yang disediakan. Oleh karena itu, kecepatan eksekusi program yang menggunakan algoritma ini juga ditentukan berdasarkan fungsi hash yang digunakan.

Pada saat melakukan pengecekan pola dengan panjang n dan teks dengan panjang m, algoritma ini akan melakukan perbandingan per karakter dari pola dan teks. Apabila karakter dari pola dan teks memiliki nilai yang sama dari hasil fungsi hash, maka pengecekan akan berlanjut pada karakter setelahnya hingga selesai. Apabila karakter dari pola dan teks memiliki nilai fungsi hash yang berbeda, maka pola akan digeser ke kanan terhadap teks. Pergeseran dapat dilakukan sebanyak (m-n) kali. Seperti yang telah disebutkan di atas, perhitungan yang terjadi pada fungsi hash akan mempengaruhi performa dari algoritma yang dibuat. Berikut ini adalah rolling hash dari algoritma Rabin-Karp yang digunakan.

$$H = c_1 a^{k-1} + c_2 a^{k-2} + c_3 a^{k-3} + \cdots + c_k a^0$$

‘a’ adalah sebuah konstanta, ‘c’ adalah karakter-karakter input, dan ‘k’ adalah banyak karakter yang ada pada *string* yang akan dibandingkan.

Sebagai contoh, kita memiliki sebuah pola ‘abc’ dan menerapkan rolling hash Rabin Karp pada pola tersebut. ‘a’ akan bernilai 26, ‘k’ menjadi 3, dan ‘c’ akan mewakili tempat di mana alfabet muncul.

Kita dapat mencari hasil dari fungsi hash terhadap ‘bcd’ dengan menggunakan hasil fungsi hash dari ‘abc’.

$$H(\text{“abc”}) = 1 \times 26^2 + 2 \times 26^1 + 3 \times 26^0$$

Untuk mendapatkan hasil fungsi ‘bcd’ kita harus menghapus ‘a’ dan menambahkan ‘d’.

$$H(\text{“}bcd\text{”}) = H(\text{“}abc\text{”}) - H(\text{“}a\text{”}) + H(\text{“}d\text{”})$$

Kita akan mengalikan H(“abc”) - H(“a”) dengan 26 sejak kita akan mengurangi term 26^2 yang saat ini dikaitkan dengan ‘a’ dan sekarang perlu dikaitkan dengan ‘b’. Pola ini akan bergeser ke kiri sehingga ‘d’ akan memiliki koefisien 26^0.

$$H(\text{“}bcd\text{”}) = ((1 \times 26^2 + 2 \times 26^1 + 3 \times 26^0) - 1 \times 26^2) \times 26 + 4 \times 26^0$$

$$H(\text{“}bcd\text{”}) = 2 \times 26^2 + 3 \times 26^1 + 4 \times 26^0$$

Pada setiap tahapan, terdapat banyak operasi yang konstan (pertambahan atau perkalian atau pengurangan) yang diselesaikan dalam waktu yang konstan juga, O(1). Ketika mengecek jika sebuah *substring* dari sebuah teks sama dengan pola yang dicari, algoritma Rabin-Karp menunjukkan bahwa keduanya memiliki hasil fungsi hash yang sama. Algoritma ini menerapkan kompleksitas O(1) pada setiap karakter dari teks sehingga jika terdapat n karakter dari teks, maka algoritma ini memiliki kompleksitas sebesar O(n).

C. Implementasi Algoritma Rabin-Karp pada Pengecekan Source Code

Algoritma Rabin-Karp akan digunakan untuk menemukan pola pada *source code* yang diberikan untuk mendeteksi plagiarisme. Pola yang digunakan pada pemeriksaan menggunakan algoritma ini adalah potongan kode yang dicurigai sama dengan atau terdapat pada sebuah *source code* tertentu. Tingkat kesamaan bisa diatur untuk menyesuaikan ketentuan plagiarisme, namun saat ini kita akan mengasumsikan pola yang sama persis dengan yang terdapat pada potongan *source code* baru akan mengindikasikan plagiarisme.

Algoritma Rabin-Karp yang digunakan pada makalah ini merupakan modifikasi dan pengembangan dari algoritma yang telah ada dan bersumber dari https://github.com/mccricardo/Rabin-Karp/blob/master/rabin_karp.py (diakses pada tanggal 24 April 2019). Algoritma dibuat dengan paradigma pemrograman berorientasi objek dengan menggunakan bahasa Python. Berikut ini adalah rincian terkait implementasi algoritma Rabin-Karp dalam mendeteksi plagiarisme pada *source code*.

A. Konstruktor Kelas RabinKarp

Pada saat melakukan inisialisasi, program akan memanggil konstruktor yang menerima dua buah parameter yaitu text dan patternSize. Parameter text digunakan untuk menerima input teks dari pengguna sedangkan parameter patternSize digunakan untuk menerima input panjang pola dari pengguna. Konstruktor akan membentuk objek yang memiliki lima buah atribut yaitu text, patternSize, hash, idxStart, dan idxEnd. Atribut text digunakan untuk menyimpan teks yang diperiksa terhadap pola. Atribut patternSize digunakan untuk menyimpan panjang pola yang akan dibandingkan. Atribut

hash digunakan untuk menyimpan nilai dari fungsi hash yang dilakukan terhadap teks. Atribut idxStart dan idxEnd digunakan untuk menyimpan batas pemeriksaan teks terhadap pola.

```
class RabinKarp:
    def __init__(self, text, patternSize):
        self.text = text
        self.patternSize = patternSize
        self.hash = 0

        for i in range (patternSize):
            self.hash += (ord(self.text[i]) - ord("a")+1)*(26**(patternSize - i -1))

        self.idxStart = 0
        self.idxEnd = patternSize
```

Gambar 5. *Screenshot* dari konstruktor kelas RabinKarp
(sumber : dokumen pribadi)

B. *Pergeseran Pengecekan*

Prosedur moveChecking digunakan untuk melakukan pergeseran teks dengan jarak batas indeks yang tetap pada saat pengecekan. Prosedur ini akan dipanggil jika ditemukan ketidakcocokan pada nilai dari fungsi hash milik pola dan milik teks yang diperiksa. Jika prosedur ini dipanggil, maka akan terjadi beberapa perubahan terhadap atribut pada objek. Pada saat digeser, karakter pertama pada teks yang dicek akan dihapus dan digantikan oleh karakter setelah indeks terakhir batas indeks pengecekan namun diletakkan di bagian paling kanan. Akibatnya nilai dari fungsi hash harus dikurangi sebanyak nilai dari fungsi hash karakter pertama tersebut. Setelah dikurangi, nilai dari fungsi akan dikali dengan 26 karena karakter-karakter yang tersisa akan mundur satu indeks. Pada saat karakter baru ditambahkan, nilai dari fungsi hash akan ditambahkan dengan nilai dari fungsi hash karakter baru. Selain itu, pergeseran menyebabkan batas indeks mulai dan akhir dari pengecekan harus digeser ke kanan sebanyak 1 langkah.

```
def moveChecking(self):
    if (self.idxEnd <= len(self.text) - 1):
        self.hash -= (ord(self.text[self.idxStart]) - ord("a")+1)*26**(self.patternSize-1)
        self.hash *= 26
        self.hash += ord(self.text[self.idxEnd])- ord("a")+1
        self.idxStart += 1
        self.idxEnd += 1
```

Gambar 6. *Screenshot* dari prosedur moveChecking
(sumber : dokumen pribadi)

C. *Pengecekan Teks*

Fungsi checkingText digunakan untuk mengembalikan teks yaitu dibatasi indeks mulai dan akhir untuk diperiksa. Fungsi ini akan dipanggil ketika dibutuhkan untuk perbandingan dengan pola yang sedang diperiksa. Fungsi ini digunakan sebagai salah satu metode dari kelas RabinKarp dan bisa dianggap sebagai sebuah *getter* dengan batas indeks untuk atribut text yang menyimpan teks yang akan diperiksa.

```
def checkingText(self):
    return self.text[self.idxStart:self.idxEnd]
```

Gambar 7. *Screenshot* dari fungsi checkingText
(sumber : dokumen pribadi)

D. *Algoritma Utama*

Algoritma utama Rabin-Karp terletak pada prosedur rabin_karp yang menerima dua buah parameter yaitu pattern dan text. Pattern digunakan untuk menampung pola yang akan dicari sedangkan text digunakan untuk menampung teks yang akan dibandingkan.

Pada saat awal dijalankan, akan dilakukan pengecekan terlebih dahulu terhadap dua buah parameter input. Jika salah satu dari parameter atau keduanya tidak memiliki nilai, maka pengecekan akan diberhentikan. Selain itu pengecekan akan berhenti jika panjang pola lebih besar dari panjang teks yang diperiksa.

Jika kedua input telah sesuai dengan ketentuan, maka akan dibentuk dua buah objek dari kelas RabinKarp untuk pola dan teks yaitu word_hash dan rolling_hash. Dengan ini, objek tersebut akan memiliki metode dan atribut yang sama karena termasuk kelas yang sama.

Setelah kedua objek terbentuk, akan dilakukan pengecekan sebanyak selisih panjang teks dan pola ditambah dengan 1. Apabila ditemukan bagian dari teks dan pattern yang memiliki nilai dari fungsi hash yang sama, maka pencarian akan berhenti dan langsung mengembalikan indeks pertama pola ditemukan pada teks. Jika tidak ditemukan, maka tidak ada nilai yang akan dikembalikan.

```
def rabin_karp(pattern, text):
    if pattern == "" or text == "":
        return None
    if len(pattern) > len(text):
        return None

    rolling_hash = RabinKarp(text, len(pattern))
    word_hash = RabinKarp(pattern, len(pattern))

    for i in range(len(text) - len(pattern) + 1):
        if rolling_hash.hash == word_hash.hash:
            if rolling_hash.checkingText() == pattern:
                return i
        rolling_hash.moveChecking()
    return None
```

Gambar 8. *Screenshot* dari prosedur rabin_karp
(sumber : dokumen pribadi)

Berdasarkan penjelasan-penjelasan sebelumnya, kita dapat merangkum dan menulis langkah-langkah yang digunakan pada saat melakukan pemeriksaan pola terhadap teks dengan menggunakan algoritma Rabin-Karp. Berikut ini adalah prosedur yang dilakukan dalam pencarian.

- Pengguna akan langsung memasukkan pola dan teks sebagai parameter pada prosedur rabin_karp.
- Jika pola dan teks yang dimasukkan sudah sesuai dengan ketentuan, maka akan dibentuk dua buah objek

dengan parameter tersebut masing-masing akan menjadi parameter bagi konstruktor objek.

- Pencarian akan dilaksanakan sebanyak selisih panjang teks dan pola ditambah 1.
- Jika nilai dari fungsi hash pola dan bagian teks yang diperiksa sama, maka akan dilakukan perbandingan antara sesama *string*.
- Jika sama, maka prosedur akan mengembalikan indeks pertama ditemukannya pola di dalam teks dan pencarian diberhentikan.
- Jika tidak sama, maka prosedur tidak akan mengembalikan nilai apapun.

E. Kasus Uji

Dalam melakukan pengujian, kita hanya akan mengambil sampel berupa potongan kode dalam satu baris saja. Untuk kasus uji pertama, kita akan menggunakan "range" sebagai pola yang akan dicari pada teks "for i in range 5".

Pola : range

Teks : for i in range 5

Berikut ini adalah langkah-langkah pencarian yang ditempuh pada pencarian di atas.

1. Pola 'range' akan dibandingkan dengan teks 'for i'. Pola 'range' memiliki nilai hash sebesar 8.252.795 sedangkan teks 'for i' memiliki nilai hash sebesar 3.016.009.
2. Pola dan teks yang diperiksa memiliki nilai hash yang berbeda akan menyebabkan prosedur moveChecking dipanggil dan akan melakukan perbandingan nilai hash 'range' dan 'or i '.
3. Perbandingan terus dilakukan sampai batas maksimal atau ditemukannya pola di dalam teks. Berikut ini adalah rincian perbandingannya.

'range' : 8.252.795 (pembanding)

'for i in range 5' : 3.016.009

'for i in range 5' : 7.127.914

'for i in range 5' : 7.105.133

'for i in range 5' : -29.131.296

'for i in range 5' : 2.994.304

'for i in range 5' : -29.080.462

'for i in range 5' : 4.316.053

'for i in range 5' : 5.285.008

'for i in range 5' : -28.929.049

'for i in range 5' : 8.252.795 (cocok)

Untuk kasus uji kedua, kita akan menggunakan pola dan teks sebagai berikut.

Pola : print

Teks : then print();

Berikut ini adalah langkah-langkah pencarian yang ditempuh pada pencarian di atas.

'print' : 7.634.452 (pembanding)

'then print();' : 9.283.808

'then print();' : 3.751.504

'then print();' : 2.488.114

'then print();' : 5.284.093

'then print();' : -28.952.832

'then print();' : 7.634.452 (cocok)

Untuk kasus uji ketiga, kita akan menggunakan pola dan teks sebagai berikut.

Pola : for i in range 3

Teks : for i in range 52

Berikut ini adalah langkah-langkah pencarian yang ditempuh pada pencarian di atas.

'for i in range 3' :11.069.783.022.442.048.541.215 (pembanding)

'for i in range 52' : 11.069.783.022.442.048.541.217

'for i in range 52' : 26.161.901.186.920.017.712.940

Pada kasus di atas, kita tidak dapat menemukan pola karena tidak ada hasil dari fungsi hash yang sesuai.

D. KESIMPULAN

Berdasarkan pengujian yang telah dilakukan, kita dapat menyimpulkan bahwa algoritma Rabin-Karp dapat digunakan untuk mendeteksi plagiarisme dalam *source code*. Algoritma Rabin-Karp mampu membandingkan pola terhadap teks dengan menggunakan fungsi hash. Fungsi hash yang dibuat mampu mengkonversi karakter menjadi sebuah nilai ASCII untuk dibandingkan antara tiap bagian teks.

References

[1] Munir, Rinaldi, Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Program Studi Teknik Informatika Institut Teknologi Bandung, 2009

[2] Putri, dkk. Examination of Document Similarity Using Rabin-Karp Algorithm.

[3] Leonardo, Brinardi. Text Documents Plagiarism Detection Using Rabin-Karp and Jaro-Winkler Distance Algorithm. 2017

[4] Ramadhani, Siti. Sistem Pencegahan Plagiarism Tugas Akhir Menggunakan Algoritma Rabin-Karp.

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, 26 April 2019

Mgs. Muhammad Riandi Ramadhan
13517080