DETEKSI SIMILARITAS DOKUMEN ILMIAH MENGGUNAKAN ALGORITMA RABIN-KARP

[1]Hermas Yuda Pamungkas , [2]Fitrianingsih
[1,2]Fakultas Teknologi Industri Universitas Gunadarma
Jl. Margonda Raya No. 100, Depok 16424, Jawa Barat
[2]fitrianingsih@staff.gunadarma.ac.id,

Abstrak

Karya ilmiah adalah salah satu hak kekayaan atas intelektual seseorang. Dalam menulis karya ilmiah tentu memiliki tantangan tertentu agar tulisan tersebut merupakan tulisan yang orisinil dan tidak menjiplak secara langsung terhadap karya tulisan orang lain. Di jaman yang serba modern ini sumber daya tulisan mudah didapatkan dimana saja melalui internet. Kemudahan tersebut membuat seseorang dapat melakukan plagiarisme dari sumber tulisan lain dengan melakukan copy-paste pada penulisannya tanpa mengubah struktur kalimat dan mencantumkan sumber tulisan tersebut. Pada penelitian ini akan mengimplementasikan algoritma Rabin – Karp dalam melakukan pendeteksian similaritas suatu dokumen artikel ilmiah dengan dokumen lainnya.Tahapan dari penelitian ini terdiri dari pemisahan baris kalimat, querying google search, preprocessing, yang terdiri dari proses tokenizing, filtering, dan stemming, k-gram, hasing, perhitungan kemiripan kalimat,dan perhitungan kemiripan dokumen input. Dokumen yang dibandingkan adalah kalimat per kalimat pada dokumen input dengan hasil pencarian kalimat tersebut pada google search. Perbandingan tersebut dengan cara membandingkan total hash dari kalimat query dengan hash hasil pencarian kalimat tersebut sehingga dihasilkan similaritas kalimat tersebut yang nantinya diakumulasikan menjadi similaritas dokumen input tersebut. Hasil penelitian ini diharapkan dapat mendeteksi plagiarisme pada sebuah dokumen.

***Kata Kunci**: Dokumen, pencocokan, string, similaritas, query*

Abstract

Twitter as one of the social media that the company uses to account for consumer complaints. Submitting Scientific work is one of the intellectual property rights of a person. In writing scientific papers certainly have certain challenges so that the writing is original writing and does not copy directly to the writings of others. In this modern era, writing resources are easily available anywhere on the internet. This convenience allows one to plagiarism from other sources of writing by copy-pasting the writing without changing the structure of the sentence and including the source of the writing. This research will implement the Rabin - Karp algorithm in detecting similarity of a scientific article document with other documents. The stages of this research consist of the separation of sentence lines, querying google search, preprocessing, which consists of the process of tokenizing, filtering, and stemming, k- gram, hasing, calculation of sentence similarity, and calculation of similarity of input documents. The document being compared is the sentence per sentence in the input document with the results of the sentence search on google search. The comparison is by comparing the total hash of the query sentence with the hash of the result of the sentence search so that the similarity of the sentence is generated which will later be accumulated into the similarity of the input document. The results of this study are expected to detect plagiarism in a document.

***Keywords** : Document, matching, string, similarity, query*

PENDAHULUAN

Penulisan suatu karya ilmiah membutuhkan data atau acuan yang sesuai dengan topik yang dibahas pada penulisan tersebut. Seiring dengan berkembangnya teknologi internet tentu mempermudah seseorang dalam mencari jurnal–jurnal dan tulisan–tulisan yang berkaitan dengan materi yang akan dicari. Adanya mesin pencarian seperti *Google* mempermudah kita dalam mencari tiap hal yang kita ketikan pada mesin pencari tersebut[1]. Data yang didapatkan menggunakan mesin pencari tersebut dapat berupa sebuah artikel, jurnal, karya ilmiah, dan lain lain[2]. Data digital tersebut dengan mudah diduplikasi kedalam penulisan suatu karya ilmiah. Kemudahan akses terhadap data yang ada di internet tersebut memperbesar kemungkinan terjadinya plagiarisme [3] terhadap tiap tulisan yang tersebar di internet. Penjiplakan atau plagiarisme berarti mencontoh atau meniru atau mencuri tulisan dan karya orang lain [4] yang kemudian diakui sebagai karangannya sendiri dengan ataupun tanpa seizin penulisnya[5].

Beberapa penelitian terkait algoritma dalam menentukan tingkat kemiripan suatu dokumen telah banyak dilakukan. Penelitian [6] Menggunakan Metode *String Matching* dengan menggunakan algoritma *Rabin – Karp.* Pada penelitian tersebut dilakukan pembuatan aplikasi pendeteksi similaritas antara 2 dokumen menggunakan algoritma *Rabin – Karp*. Pada penelitian [7] peneliti melakukan percobaan dimana dapat disimpulkan bahwa jumlah dari K-gram menentukan similaritas dari dokumen yang sedang diuji. Pada penelitian [8] , peneliti mengimplementasikan tahapan *preprocessing* pada metode Levensthein Distance untuk mempengaruhi nilai ke-miripan dokumen dan waktu proses perhitungan tersebut. Pada penelitian [9] dilakukan pengimplementasian algoritma Rabin–Karp untuk mendeteksi tingkat ke-miripan dari source code pada pem-rogramanan lisp menggunakan perhitungan *Dice Similarity Coeficient.* Pada jurnal [10] menimplementasikan algoritma *stemming* Nazief– Adriani pada pendeteksian kemiripan dokumen berbahasa indonesia yang dida-patkan kesimpulan bahwa *stemming* mem-pengaruhi hasil kemiripan pendeteksian similaritas dokumen tersebut.

Pada penelitian ini peneliti akan mengimplementasikan penggunaan algoritma Rabin-Karp melalui metode pencocokan string (*String Matching*) untuk menentukan indeks similaritas dari sebuah dokumen berbentuk artikel ilmiah berbahasa Indonesia pada hasil pencarian *Google*. Pengambilan data dari hasil pencarian mesin pencari *Google* menggunakan teknik *crawling*. Proses *stemming* menggunakan algoritma Nazief – Adriani untuk dokumen berbahasa Indonesia dan proses *k-gram* dan *hashing* pada sebuah dokumen input dan hasil pencarian kalimat tersebut. Implementasi algoritma ini diharapkan dapat mendeteksi plagiarisme pada sebuah dokumen.

METODE PENELITIAN

Tahapan awal dimulai dari pembacaan dokumen sumber yang selanjut-nya akan dipecah menjadi perkalimat pada tahap pemisahan kalimat, lalu dari kalimat tersebut akan dijadikan *query* untuk pencarian *google.* Dari data tersebut akan dilanjutkan proses *preprocessing* yaitu proses *Tokenizing, Filtering, dan Stemming.* Dari hasil *preprocessing* akan dilanjutkan dengan proses *k-gram* yang selanjutnya akan dilakukan proses *hasing.* Kemudian dari hasil *hasing* tersebut akan dihitung kemiripan kalimat dengan hasil pencarian google dan dirata–ratakan sehingga menghasilkan hasil kemiripan keseluruhan dokumen.

Dokumen Input

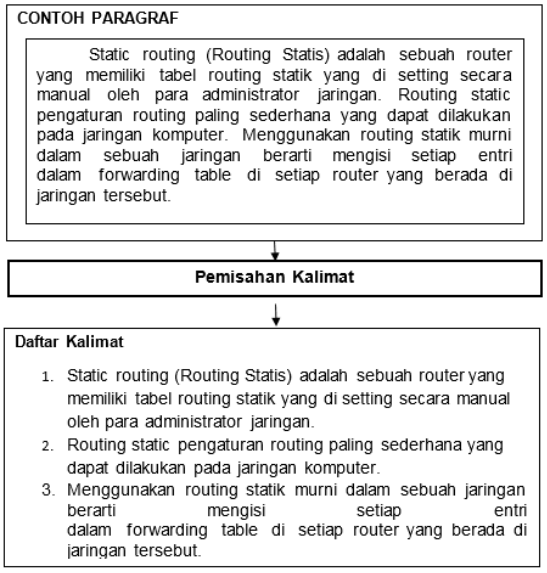
Dokumen input yang digunakan adalah dokumen yang akan dilakukan analisa kemiripan dokumen. Segala konten yang di inputkan ke dalam sistem akan di analisa perkalimat dari konten tersebut. Kalimat per kalimat akan dipisahkan dari dokumen tersebut pada langkah selanjutnya. Setelah itu, per kalimat tersebut akan dilakukan tahapan selanjutnya yaitu *querying google search.*

Pemisahan Kalimat

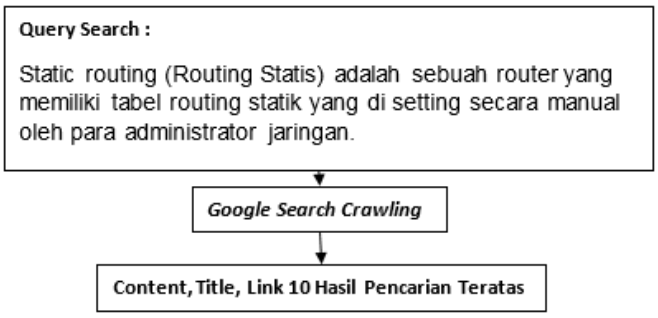
Pada tahap ini akan dilakukan pemisahan kalimat per kalimat dari sebuah dokumen input. Pemisahan ini dilakukan dari awal kalimat hingga akhir kalimat yang ditandai oleh titik. Teknik pemisahan ini menggunakan *pattern regex* pada java sebagai indikator dari kalimat tersebut. Dari sebuah teks panjang dipotong menggunakan *regex* sehingga dihasilkan kumpulan kalimat dari *string* tersebut yang nantinya digunakan dalam *crawling* untuk mendapatkan hasil yang similar dengan tiap kalimat pada kumpulan kalimat tersebut. Pada method diatas kalimat akan dipisahkan berdasarkan titik, koma, tanda seru, tanda tanya, dan titik dua. Hasil dari *method* tersebut berupa *list* yang berisi kumpulan kalimat dari sebuah dokumen input tersebut. Gambar 1 merupakan tahapan pemisahan kalimat.

Querying Google Search

Dari hasil kalimat yang sebelumnya sudah dipisahkan dari tiap paragraf pada dokumen input, dilakukan kueri pencarian pada mesin pencari *google*. Dari hasil pencarian tersebut dilakukan teknik *scrapping* pada halaman html *google* tersebut sehingga hanya didapatkan 10 pencarian teratas terhadap kalimat tersebut. Daftar pencarian tersebut dimasukkan kedalam sebuah *list object* pada java yang terdiri atas judul tulisan, link/url sumber tulisan, dan konten tulisan yang terdapat pada hasil pencarian tersebut seperti dapat dilihat pada ilustrasi Gambar 2.



Gambar 1. Ilustrasi Pemenggalan Kalimat pada Paragraf



Gambar 2. Ilustrasi Pencarian Kalimat *Query*

Pada sebuah pencarian pada *google* data ditampilkan dalam format *html* dengan tag dan *class* tertentu. Untuk mendapatkan data terlebih dahulu harus dilakukan pengambilan konten dari hasil pencarian tersebut menggunakan teknik *crawling.*

Pada Gambar 3 ditampilkan salah satu data hasil pencarian pada google dengan format *html*. Dari hasil pencarian tersebut memungkinkan untuk mendapatkan title, content,dan link sumber data tersebut. Dengan menggunakan library JSoup seluruh file html pada halaman *google* hasil pencarian kalimat input dapat diekstrak menjadi komponen tersebut.

Gambar 3. Contoh Hasil Pencarian pada *Google*

Preprocessing

Preprocessing adalah tahap dimana sebuah kalimat di lakukan pemilihan kata atau *term* yang digunakan dalam proses perhitungan kemiripan sehingga hanya tersisa kata–kata yang memiliki makna dalam kalimat tersebut. Kalimat dari dokumen imput dan hasil pencarian kalimat tersebut akan dilakukan proses *preprocessing* sebelum dilanjutkan pada proses perhitungan kemiripan. Proses ini terdiri atas 3 proses yaitu *Tokennize,Filtering*, dan *Stemming.*

1) **Tokenizing.** Melakukan pemecahan kalimat menjadi per-kata yang di tampung kedalam bentuk List pada java. Dalam melakukan proses ini dibantu menggunakan library OpenNLP yang merupakan sebuah library pada java yang digunakan dalam pemrosesan natural language processing berbasis teks. OpenNLP memiliki salah satu fitur dengan nama tokenization yang menyediakan 3 klasifikasi yaitu Simple Tokenizer, Whitespace Tokenizer, dan Tokenizer ME. Pada proses ini klasifikasi yang digunakan adalah Simpletokenizer yang hanya memisahkan kata per kata berdasarkan spasi dan kelas karakter. Pada tahap ini juga dilakukan pengilangan karakter khusus sehingga yang akan dihasilkan hanyalah alfabet dan angka. Kalimat dari dokumen beserta 10 hasil pencarian kalimat tersebut akan dilakukan tokenisasi untuk dilanjutkan pada tahap filtering.

2) **Filtering.** Setelah dilakukan proses tokenizing, list perkata dari kalimat tersebut akan disaring pada proses ini untuk menghilangkan kata– kata ataupun simbol yang dianggap tidak penting pada kalimat tersebut. Kata–kata yang dihi-

langkan pada proses ini adalah kata – kata *stopwords* atau kumpulan kata– kata yang tidak memiliki makna atau dianggap kurang penting dalam sebuah kalimat sehingga hanya tersisa kata– kata yang dianggap memiliki makna pada kalimat tersebut pada hasil proses *filtering*.

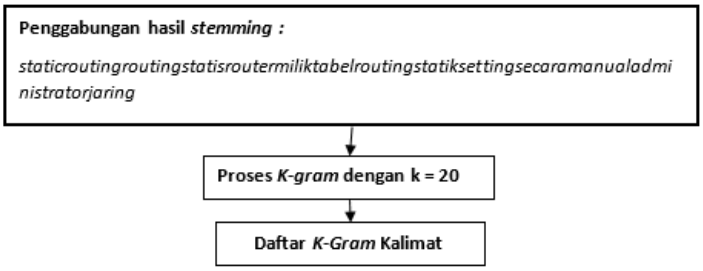
Langkah-langkah dari proses *filtering* pada penelitian ini adalah sebagai berikut:

1. Filterisasi kata akan dilakukan sebanyak jumlah kata pada hasil tokenisasi.
2. Fungsi filetrisasi akan melakukan pengecekan pada daftar *stopwords* bahasa Indonesia, apabila terdapat kata pada daftar tersebut maka kata tersebut akan dihapus pada daftar hasil filterisasi. Jika tidak maka perulangan akan berlanjut pada kata selanjutnya
3. Perulangan akan terhenti apabila sudah tidak ada kata lagi yang akan diperiksa.

3) **Stemming.** Proses mengubah kata menjadi kata dasar dari kata tersebut. proses ini akan menghilangkan semua imbuhan pada sebuah kata sehingga dihasilkkan hanya kata dasarnya saja. Sehingga hasil dari kalimat yang di lakukan proses perhitungan kemiripan lebih optimal.Pada tahap ini hasil dari tahap filtering akan diubah menjadi kata dasarnya sehingga yang diproses pada perhitungan kemiripan kalimat hanyalah daftar kata dasar yang sudah dihilangkan imbuhannya dari awal hingga akhir. Algoritma *stemming* yang digunakan pada penelitian ini adalah algoritma *stemming* Nazief–Adriani untuk dokumen berbahasa Indonesia. Algoritma Nazief – Adriani didasarkan pada aturan morfologi Bahasa Indonesia yang dikumpulkan menjadi satu grup dan di enkapsulasi pada imbuhan/*affixes* yang diperbolehkan (*allowed affixes)* dan imbuhan/*affixes* yang tidak diperbolehkan (*disallowed affixes).*

K-Gram

Pada tahap ini hasil dari daftar kata pada proses *stemming* akan dibentuk suatu string berurutan yang nantinya akan dipotong berdasarkan jumlah k-gram yang nanti akan dilanjutkan pada proses *hashing*. Metode *k-gram* sendiri adalah proses pemotongan sejumlah karakter pada sebuah string dengan jumlah tertentu. Berikut adalah ilustrasi dari proses *k-gram* dari sebuah kalimat yang akan diproses pada tahap sebelumnya. Gambar 4 merupakan ilustrasi dari *k-gram.*



Gambar 4. Ilustrasi Proses *Stemming*

Tabel 1. Contoh Hasil Proses *Hashing*

No.	Hasil K-Gram	Hasil Hash
1.	staticroutingrouting	2731643090344028
2.	taticroutingroutings	2690960813048380
3.	aticroutingroutingst	2392181994929516
4.	ticroutingroutingsta	2710269105507052
5.	icroutingroutingstat	2488723457222876
6.	croutingroutingstati	2430036963848860

Hashing

Hasing yang digunakan pada penelitian ini adalah *rolling hash* dimana *list* k-gram pada tahap sebelumnya akan di hash sebelum diperiksa kemiripan karakternya dengan hasil pencarian kalimat tersebut. Dari hasil *k-gram* sebelumnya akan dihasilkan hasil *hash* yang unik sehingga dalam perbandingan kemiripan kalimat lebih akurat. Tabel 1 merupakan contoh hasil dari proses *hashing* dari sebuah daftar *string* yang dihasilkan pada tahap *k-gram* sebelumnya.

Perhitungan Kemiripan Kalimat

Sebelum dilakukan perhitungan terlebih dahulu dilakukan pengecekan *hash* antara *hash list* kalimat uji dengan 10 *content* hasil pencarian kalimat tersebut pada *google search*. Pengecekan kemiripan tiap object *hash* disini menggunakan kelas java stream() dengan menggunakan fungsi *contains*. Penggunaan fungsi tersebut mengurangi beban dalam perhitungan kemiripan dengan kompleksitas O(n) dibandingkan *nested loop* dengan kompleksitas O(n*n). Dari hasil pencarian tersebut akan didapatkan daftar *hash* yang sama dari *list hash* kalimat *query* dengan *hash* hasil pencarian kalimat tersebut yang selanjutnya akan ditampung pada *list* match untuk didapatkan banyaknya *list* yang sama tersebut dengan method *size*().

```
List<Words> match = searchData
        .stream()
        .filter(queries::contains)
        .collect(Collectors.toList());
```

Dari jumlah *hash* yang sama tersebut dapat dihitung persentase kemiripan antara

kalimat *query* dengan hasil pencarian kalimat tersebut pada mesin pencari *google*. Perhitungan yang dilakukan adalah dengan membagi keseluruhan total *hash* yang sama dengan total *hash* pada kalimat dikalikan dengan 100 untuk menghasilkan persentase kemiripan kalimat menggunakan persamaan 1.

$$Similaritas\ Kalimat = \frac{\sum Total\ Hash\ Sama}{\sum Total\ Hash\ Query} \times 100\% \quad (1)$$

Perhitungan Kemiripan Keseluruhan Dokumen

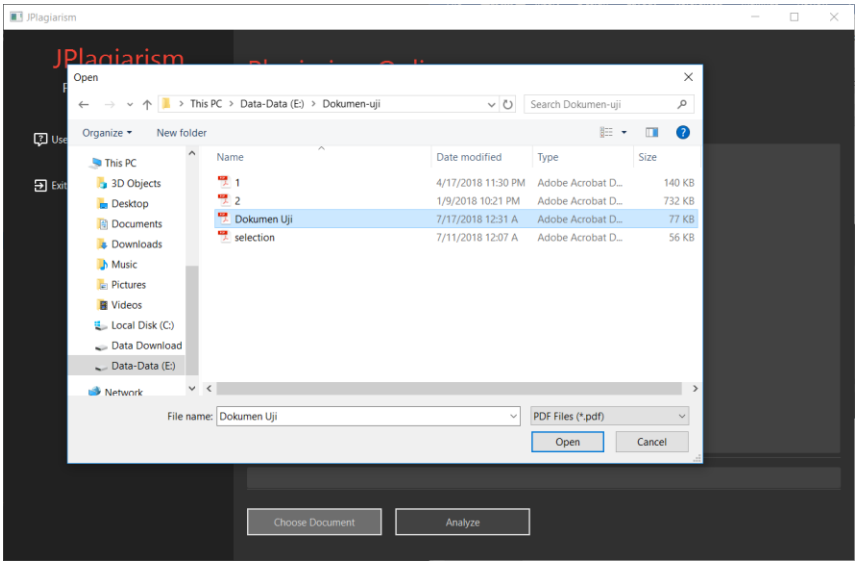
Perhitungan kemiripan keseluruhan dokumen dihitung dari total persentase dari semua kalimat yang telah dilakukan pengecekan kemiripan dibagi total kalimat dari dokumen tersebut. Penentuan apakah dokumen tersebut termasuk plagiarisme berat atau tidak dirujuk pada persentase kemiripan (Tabel 2.3. iThenticate, 2018)

Ttotal similaritas dokumen maka dilakukan penjumlahan total persentase similaritas tiap kalimat pada dokumen tersebut yang selanjutnya akan dibagi dengan jumlah kalimat yang terdapat pada dokumen tersebut seperti pada persamaan 2.

$$Similaritas\ Dokumen = \frac{\sum Total\ Similaritas\ Kalimat}{\sum Total\ Kalimat\ Pada\ Dokumen} \quad (2)$$

HASIL DAN PEMBAHASAN

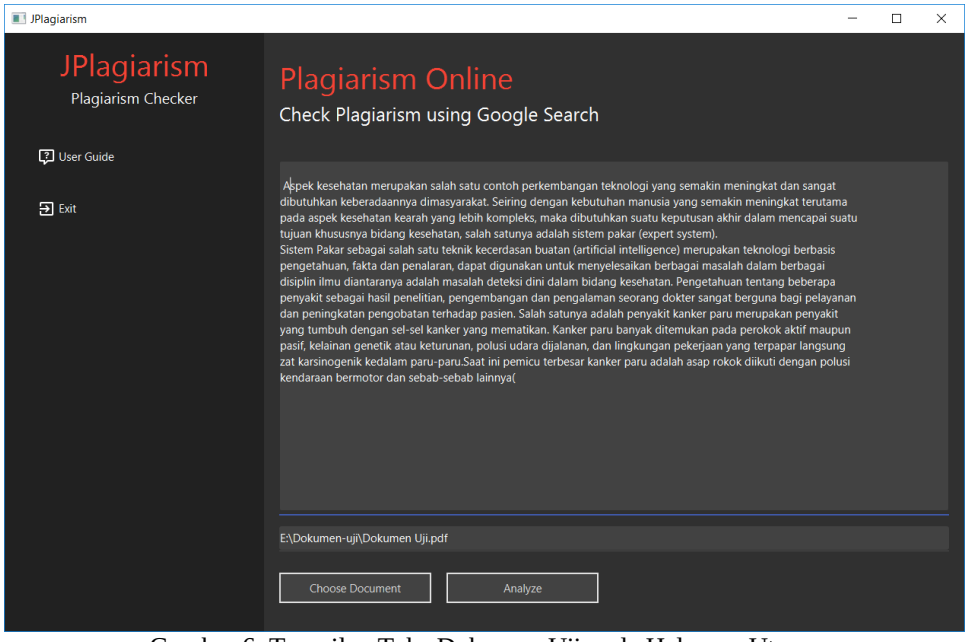
Langkah awal dilakukan penginputan dokumen input berupa file pdf untuk menganalisa dokumen. Pada kasus ini dokumen input bernama "Dokumen Uji.pdf" akan dilakukan pengujian plagiarisme. Pilih dokumen lalu tekan open untuk membuka file tersebut seperti dapat dilihat pada Gambar 5.



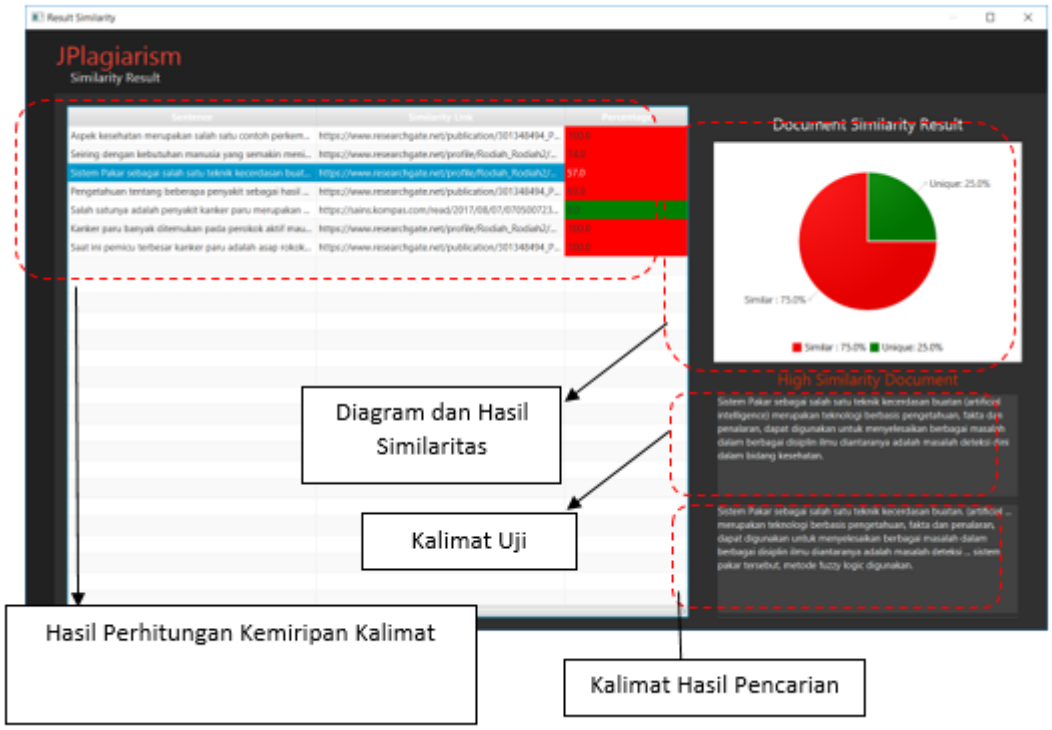
Gambar 5. Pemilihan File Uji

Setelah file terpilih maka akan tampil teks hasil parsing file pdf tersebut menjadi teks dengan library iText. Jika teks sudah terisi maka pencarian similaritas akan dapat dilakukan. Tekan *analyze* untuk melanjutkan pada tahap pencarian similaritas. Pada tahap pencarian similaritas ini waktu yang dibutuhkan dalam pencarian bergantung pada koneksi internet dan banyaknya kalimat pada dokumen input tersebut. Semakin banyak kalimat dalam dokumen tersebut maka akan semakin lama proses perhitungannya. Gambar 6 merupakan tampilan setelah diinputkan file uji.

Setelah proses pencarian kalimat, *preprocessing*, hingga tahap perhitungan selesai maka akan tampil halaman report pada Gambar 7, dimana akan ditampilkan dokumen input yang sudah dipisahkan perkalimat menggunakan *regex* dengan url yang mengandung kalimat tersebut serta nilai similaritas. Seluruh kalimat pada dokumen input dan hasil pencarian similaritasnya akan ditampilkan pada tabel pada halaman tersebut. Untuk dapat melihat *content* dari hasil *crawl* dapat menekan baris pada tabel tersebut lalu akan tampil pada textfield bagian bawah chart dimana textfield pertama menampilkan kalimat dari dokumen input yang digunakan sebagai kalimat *query*, dan *textfield* kedua sebagai content dari hasil pencarian dengan indeks similaritas tertinggi. Hasil keseluruhan similaritas kalimat pada dokumen tersebut ditampilkan dalam bentuk *pie chart* dan kesimpulan akhir dari kalkulasi persentase similaritas dokumen tersebut dengan teks dimana kesimpulan tersebut didapatkan berdasarkan penilaian indeks similaritas menurut iThenticate.



Gambar 6. Tampilan Teks Dokumen Uji pada Halaman Utama



Gambar 7. Tampilan Hasil Akhir Perhitungan Kemiripan Dokumen

KESIMPULAN DAN SARAN

Beberapa kesimpulan yang dapat diambil pada penelitian ini antara lain : Pemisahan kalimat pada dokumen input menggunakan *pattern regex* telah berhasil dilakukan dengan baik.Pencarian kalimat pada dokumen input pada mesin pencari *google* dan *crawling* data dari hasil pencarian tersebut telah berhasil dilakukan. Proses *Stemming* untuk mendapatkan kata dasar dari sebuah kata dengan menggunakan algoritma *stemming* untuk dokumen berbahasa Indoneisa Nazief–Adriani pada sebuah kalimat input maupun konten hasil pencarian kalimat tersebut telah berhasil dilakukan dengan baik. Penelitian ini berhasil melakukan pembentukan *k-gram* dan *hashing* dari *string output* tahap preprocessing telah berhasil dilakukan dengan baik. Proses perhitungan jumlah *hash* yang sama pada kalimat *query* dan 10 konten hasil pencarian pada mesin pencari *google* telah berhasil dilakukan dengan baik. Penelitian ini berhasil membentuk aplikasi pendeteksi similaritas tulisan ilmiah menggunakan sumber hasil pencarian mesin pencari *google.*

Pengembangan lebih lanjut yang dapat dilakukan pada penelitian selanjutnya antara lain dengan mengimplementasikan algoritma yang dapat mempertimbangkan posisi dan urutan kata, sinonim kata dari sebuah kalimat, dan kemiripan semantik dari sebuah dokumen input sehingga menambah nilai akurasi dari similaritas dokumen dengan dokumen pembanding. Selain itu, dalam teknik *crawling*

dapat dikembangkan dengan melakukan *crawling* dari tiap website yang diketahui memiliki similaritas dengan kalimat input secara utuh, tidak hanya dari hasil pencarian *google search*.

DAFTAR PUSTAKA

[1] S. S. Rajmohammod, P. P. Arun, dan G. P. Dnyandeo, "Web crawler : Extracting the web data", *International Engineering Research Journal (IERJ)*, vol. 1, no. 8, hal. 629–632, 2015.

[2] N. Wubbeling, "The South African Journal of Science-iThenticate", Academy of Science of Scouth Africa (ASSAF), 2016.

[3] R. Janani dan S. Vijayarani, "An efficient text pattern matching algorithm for retrieving information from desktop", *Indian Journal of Science and Technology*, vol. 9, no. 43, 2016.

[4] C. D. Manning, P. Raghavan, dan H. Schutze, *An introduction to information retrieval*. Cambridge: Cambridge University Press, 2008.

[5] B. Gipp dan N. Meuschke, "Citation pattern matching algorithms for citation-based plagiarism detection: greedy citation tiling, citation chunking and longest common Citation Sequence", dalam Proceedings of the 11th ACM Symposium on Document Engineering - DocEng '11, 2011, hal. 249–258.

[6] A. Prastyanti, "Sistem deteksi kemiripan kata pada dua dokumen menggunakan algoritma Rabin-Karp", Skripsi Sarjana, Universitas Diponegoro, Semarang, 2014.

[7] A. P. U. Siahaan, Mesran, R. Rahim, dan D. Siregar, "K-Gram as a determinant of plagiarism level in Rabin-Karp algorithm", *International Journal of Scientific & Technology Research*, vol. 6, no. 7, 2017.

[8] N. H. Ariyani, Sutardi, dan R. Ramadhan, "Aplikasi pendeteksi kemiripan isi teks dokumen menggunakan metode Levenshtein Distance", *SemanTIK*, vol. 2, no. 1, hal. 279–286, 2016.

[9] A. A. Wiguna dan I. Rizqa, "Pemanfaatan algoritma Rabin-Karp untuk mengetahui tingkat kemiripan dari *source code* pada pemrograman Lisp", Skripsi Sarjana, Universitas Dian Nuswantoro, Semarang, 2016.

[10] A. Firdaus, Ernawati, dan A. Vatresia, "Aplikasi pendeteksi kemiripan pada dokumen teks menggunakan algoritma Nazief & Andriani dan metode *Cosine Similirity*", *Jurnal Teknologi Informasi*, vol. 10, no. 1, hal. 96–109, 2014.