MENGUKUR TINGKAT *SIMILARITY* ANTARA *SOURCE CODE* SATU DAN LAINNYA PADA BAHASA PEMROGRAMAN JAVA MENGGUNAKAN ALGORITMA *LEVENSHTEIN DISTANCE*

SKRIPSI

Oleh :
MOH. ALAMSYAH ADI KARTANEGARA
NIM. 15650068

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2022

MENGUKUR TINGKAT *SIMILARITY* ANTARA *SOURCE CODE* SATU DAN LAINNYA PADA BAHASA PEMROGRAMAN JAVA MENGGUNAKAN ALGORITMA *LEVENSHTEIN DISTANCE*

SKRIPSI

Diajukan kepada:
Universitas Islam Negeri Maulana Malik Ibrahim Malang
Untuk memenuhi Salah Satu Persyaratan dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh :
MOH. ALAMSYAH ADI KARTANEGARA
NIM. 15650068

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2022

HALAMAN PERSETUJUAN

MENGUKUR TINGKAT *SIMILARITY* ANTARA *SOURCE CODE* SATU DAN LAINNYA PADA BAHASA PEMROGRAMAN JAVA MENGGUNAKAN ALGORITMA *LEVENSHTEIN DISTANCE*

SKRIPSI

Oleh :
MOH. ALAMSYAH ADI KARTANEGARA
NIM. 15650068

Telah Diperiksa dan Disetujui Untuk Diuji
Tanggal: 21 Juni 2022

Dosen Pembimbing I	Dosen Pembimbing II
Prof. Dr. Suhartono, M.Kom	Roro Inda Melani, M. T., M. Sc
NIP. 19680519 200312 1 001	NIP. 19780925 200501 2 008

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Irachmi Kurniawan ST., M.MT ., IPM
NIP. 19771020 200912 1 001

HALAMAN PENGESAHAN

MENGUKUR TINGKAT *SIMILARITY* ANTARA *SOURCE CODE* SATU DAN LAINNYA PADA BAHASA PEMROGRAMAN JAVA MENGGUNAKAN ALGORITMA *LEVENSHTEIN DISTANCE*

SKRIPSI

Oleh :
MOH. ALAMSYAH ADI KARTANEGARA
NIM. 15650068

Telah Dipertahankan di Depan Dewan Penguji Skripsi
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Tanggal : 21 Juni 2022

Susunan Dewan Penguji

Penguji Utama	:	Dr. Cahyo Crysdian, MCS NIP. 19740424 200901 1 008	( )
Ketua Penguji	:	A'la Syauqi, M. Kom NIP. 19771201 200801 1 007	( )
Sekretaris Penguji	:	Prof. Dr. Suhartono, M. Kom NIP. 19680519 200312 1 001	( )
Anggota Penguji	:	Roro Inda Melani, M. T., M. Sc NIP. 19780925 200501 2 008	( )

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Fachrul Kurniawan ST., M.MT ., IPM
NIP. 19771020 200912 1 001

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan di bawah ini:

Nama	: Moh. Alamsyah Adi Kartanegara
NIM	: 15650068
Fakultas / Jurusan	: Sains dan Teknologi / Teknik Informatika
Judul Skripsi	: Mengukur Tingkat *Similarity* Antara *Source Code* Satu dan Lainnya pada Bahasa Pemrograman Java Menggunakan Algoritma *Levenshtein Distance*

Menyatakan dengan sebenarnya bahwa Skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan, atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka.

Apabila dikemudian hari terbukti atau dapat dibuktikan skripsi ini merupakan hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 21 Juni 2022
Yang membuat pernyataan,

Moh. Alamsyah Adi Kartanegara
NIM.15650068

MOTTO

Barangkali hikmah suatu kejadian tak selalu datang setelahnya, bisa jadi kita paham bahwa semua ketentuan itu baik setelah hitungan minggu, bulan, tahun, pun bersemayam sepanjang hidup.

HALAMAN PERSEMBAHAN

Karya tulis ini penulis persembahkan untuk orang-orang terkasih yang selalu mendukung penulis di tiap keadaan. Semoga karya tulis ini menjadi batu loncatan untuk karya-karya setelahnya, dan bisa jadi pengingat kepada penulis sendiri untuk selalu mempunyai integritas dan kejujuran intelektual ke depannya.

KATA PENGANTAR

Assalamu'alaikum Warrahmatullahi Wabarakatuhu

Alhamdulillah rabbil 'alamiin, segala puji bagi Allah SWT, yang telah memberikan karunia, rahmat dan hidayah-Nya. Sehingga memberikan kemudahan dalam proses penyusunan skripsi dengan judul "**Mengukur Tingkat *Similarity* Antara *Source Code* Satu dan Lainnya pada Bahasa Pemrograman Java Menggunakan Algoritma *Levenshtein Distance***" hingga selesai. Salawat serta salam semoga senantiasa tersampaikan kepada Nabi Muhammad SAW yang memberikan syafaat dari zaman jahiliyah menuju zaman yang penuh berkah.

Penulis menyadari banyak keterbatasan yang penulis miliki, sehingga banyak pihak yang telah memberikan bantuan baik moril maupun materil dalam proses menyelesaikan penelitian ini. Maka dari itu dengan segenap kerendahan hati penulis mengucapkan terima kasih kepada:

1. Prof. Dr. M. Zainuddin, MA selaku Rektor UIN Maulana Malik Ibrahim Malang.
2. Dr. Sri Harini M.Si selaku Dekan Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
3. Dr. Fachrul Kurniawan ST., M.MT., IPM selaku Ketua Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
4. Prof. Dr. Suhartono, M. Kom selaku dosen pembimbing pertama dan Roro Inda Melani, M. T., M.Sc selaku dosen pembimbing kedua yang yang selalu sabar, perhatian serta pemberi solusi setiap kebingungan hingga penelitian ini dapat terselesaikan dengan lancar.
5. Seluruh dosen dan staf jurusan Teknik Informatikan Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang yang telah memberikan ilmu dan pengalaman yang bermanfaat.
6. Kedua Orang Tua dan seluruh keluarga besar yang senantiasa mendukung dan mendoakan penulis.

7. Rekan-rekan dan sahabat seperjuangan Jurusan Teknik Informatika 2015 Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
8. Senior-senior maupun rekan-rekan Relawan Nusantara dan Dompet Dhuafa Volunteer yang selalu memberikan teladan nyata dalam beberapa tahun terakhir.

Penulis menyadari dalam karya ini masih banyak kekurangan. Oleh karena itu penulis selalu menerima segala kritik dan saran dari pembaca. Semoga karya ini dapat bermanfaat dan dipergunakan mestinya bagi seluruh pihak.

Wassalamu'alaikum Warrahmatullahi Wabarakatuhu

Malang, 21 Juni 2022

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Kartanegara, Moh. Alamsyah Adi. 2022. **Mengukur Tingkat *Similarity* Antara *Source Code* Satu dan Lainnya Pada Bahasa Pemrograman Java Menggunakan Algoritma *Levenshtein Distance***. Skripsi. Jurusan Teknik Informatika Fakultas Sains dan Teknologi. Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Prof. Dr. Suhartono, M. Kom. (II) Roro Inda Melani, M.T, M. Sc.

Kata Kunci: *Java, Levenshtein Distance, Pemrograman, Plagiarisme , Source Code.*

Plagiarisme merupakan tindakan mengambil karya orang lain tanpa mencantumkan sumber aslinya, yang di mana hal tersebut terlarang, terlebih lagi di dunia akademis. Di dunia pemrograman, *source code* merupakan hal yang penting untuk membangun sebuah aplikasi. Algoritma *Levenshtein Distance* yang fungsi utamanya untuk mengukur jarak antara dua *string*, digunakan untuk mengukur persentase *similarity* pada *source code* bahasa pemrograman Java. Pada penelitian ini, data uji *source code* yang digunakan sebanyak 20 data, di mana sistem membandingkan satu *source code* kepada banyak *source code*. Hal ini dilakukan melalui 4 skenario uji coba, yaitu menggunakan *Levenshtein Distance* autentik, *Levenshtein Distance* ditambah *preprocessing casefolding, Levenshtein Distance* ditambah *preprocessing filtering*, dan yang terakhir *Levenshtein Distance* ditambah *preprocessing case folding* dan *filtering*. Dari keempat skenario tersebut, didapatkan hasil paling signifikan pada skenario kedua, yaitu berupa penghitungan dengan algoritma *levenshtein distance* ditambah tahapan *preprocessing case folding*, yaitu *accuracy* sebesar 84,2%, *precision* 89,4%, *recall* 87,6%, dan *F-1 Score* 87,9%.

ABSTRACT

Kartanegara, Moh. Alamsyah Adi. 2022. **Measuring the Level of Similarity Between One Source Code and Another in The Java Programming Language Using The Levenshtein Distance Algorithm** . Thesis. Department of Informatics Engineering Faculty of Science and Technology. Maulana Malik Ibrahim State Islamic University Malang. Supervisor: (I) Prof. Dr. Suhartono, M. Kom. (II) Roro Inda Melani, M. T, M. Sc.

Keywords: *Java, Levenshtein Distance , Programming, Plagiarism , Source Code.*

Plagiarism is the act of taking someone else's work without acknowledging the original source, which is forbidden, especially in academia. In the world of programming, source code is an important thing to build an application. The Levenshtein Distance algorithm whose main function is to measure the distance between two strings , is used to measure the percentage of similarity in the Java programming language source code . In this study, the source code test data used as many as 20 data, where the system compares one source code to many source codes . This is done through 4 test scenarios, namely using authentic Levenshtein Distance, Levenshtein Distance plus preprocessing casefolding, Levenshtein Distance plus preprocessing filtering, and the last Levenshtein Distance plus preprocessing case folding and filtering. Of the four scenarios, the most significant results were obtained in the second scenario, namely in the form of calculations using the Levenshtein distance algorithm plus the preprocessing case folding stage , namely 84.2% accuracy , 89.4% precision , 87.6% recall , and F-1 Score 87.9%.

الملخص

كرتانغرا، محمد علام شاه أدي . 22. 2. **قياس مستوى التشابه بين شفرة مصدر وأخرى في لغة برمجة جافا باستخدام خوارزمية "Levenshtein Distance"**. بحث جامعى. قسم المعلوماتية. كلية العلوم والتكنولوجيا. جامعة مولانا مالك إبراهيم الإسلامية الحكومية بمالانج. المشرف (1) الدكتور جامعى سوهرتنا الماجيستير (2) رورو إيندا ميلانى الماجيستر

الكلمات المفتاحية: *Java، Levenshtein Distance، Programming، كود المصدر، الانتحال.*

السرقة الأدبية هي أخذ عمل شخص آخر دون الاعتراف بالمصدر الأصلي ، وهو أمر ممنوع خاصة في الأوساط الأكاديمية .في عالم البرمجة ، يعد *الكود المصدري* أمرًا مهمًا لبناء تطبيق .تُستخدم خوارزمية *Levenshtein Distance* التي تتمثل وظيفتها الرئيسية في قياس المسافة بين سلسلتين ، في قياس النسبة المئوية *للتشابه* في *كود مصدر* لغة برمجة Java . في هذه الدراسة ، استخدمت بيانات اختبار *كود المصدر* ما يصل إلى 20 بيانات ، حيث يقارن النظام *كود مصدر واحد* بالعديد من *أكواد المصدر* .يتم ذلك من خلال 4 سيناريوهات اختبار ، وهي استخدام Levenshtein *Distance* الأصيلة ، و *Levenshtein Distance مسبقة المعالجة* ، و *Casefolding* بالإضافة إلى *Levenshtein Distance* بالإضافة إلى حالة المعالجة *المسبقة قابلة للطي* والتصفية .من بين *Levenshtein* إلى *تصفية المعالجة المسبقة* ، وآخر *مسافة* تم الحصول على أهم النتائج في السيناريو الثاني ، أي في شكل حسابات باستخدام خوارزمية *المسافة* ، السيناريوهات الأربعة *بالإضافة إلى مرحلة طي علبة المعالجة المسبقة* ، وهي *دقة* 84.2٪ ، *دقة* 89.4٪ ، *استرجاع* 87.6٪ ، و *Levenshtein F- 1* 87.9 *النتيجة*/..

BAB I

PENDAHULUAN

1.1 Latar Belakang

Menurut Kamus Besar Bahasa Indonesia (KBBI) , plagiarisme atau yang lebih dikenal dengan plagiat adalah pengambilan karangan (pendapat dan sebagainya) orang lain dan menjadikannya seolah-olah karangan (pendapat) sendiri. Di dalam Peraturan Menteri Pendidikan Nasional Republik Indonesia No. 17 Tahun 2010, pada pasal 1 mendefinisikan bahwa plagiat adalah perbuatan secara sengaja atau 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.

Di dalam buku yang berjudul Algoritma dan Pemrograman Menggunakan Java, program adalah kumpulan instruksi yang digunakan untuk mengatur komputer agar melakukan suatu tindakan tertentu. Untuk membuat suatu program tertentu, maka harus mengikuti kaidah atau tata aturan yang berlaku di tiap-tiap bahasa pemrograman. Bahasa pemrograman saat ini sudah sangat banyak, misalnya saja Java, PHP, C++, dan contoh-contoh lainnya.

Di dalam ranah informatika, *source code* merupakan suatu hal yang sangat penting untuk para *programmer*, dikarenakan ini adalah bagian utama untuk membangun sebuah sistem informasi ataupun aplikasi. Dalam membangun aplikasi, *programmer* memiliki banyak cara yang mempengaruhinya dalam penulisan kode program, seperti bahasa yang digunakan, *library*, dan hal-hal

lainnya. Dengan banyaknya faktor tersebut, tentu kesamaan kode program antara satu dan lainnya sangat minim terjadi. Namun, akhir-akhir ini sering kita jumpai kesamaan antara kode program satu dan lainnya jika hal itu masih dalam skala yang kecil, hal ini dikarenakan banyak faktor tentu saja.

Di dalam Islam sendiri, hal-hal seperti mengambil hak milik orang tanpa persetujuannya dan melanggar aturan sudah diatur di dalam Al-Qur'an, *hadits,* maupun perkataan-perkataan ulama.

يَا أَيُّهَا الَّذِينَ آمَنُوا لَا تَأْكُلُوا أَمْوَالَكُمْ بَيْنَكُمْ بِالْبَاطِلِ إِلَّا أَنْ تَكُونَ تِجَارَةً عَنْ تَرَاضٍ مِنْكُمْ ۚ وَلَا تَقْتُلُوا أَنْفُسَكُمْ ۚ إِنَّ اللَّهَ كَانَ بِكُمْ رَحِيمًا

Hai orang-orang yang beriman, janganlah kamu saling memakan harta sesamamu dengan jalan yang batil, kecuali dengan jalan perniagaan yang berlaku dengan suka sama-suka di antara kamu. Dan janganlah kamu membunuh dirimu; sesungguhnya Allah adalah Maha Penyayang kepadamu. (Q.S An-Nisa : 29)

Berdasarkan ayat quran di atas, sebagai muslim kita dilarang untuk mengambil barang-barang ataupun kekayaan intelektual yang bukan milik kita dan menganggapnya sebagai karya pribadi tanpa ada campur tangan dari orang lain. Hal ini juga bertentangan dengan prinsip seorang muslim yang dituntut untuk terus belajar dan bersikap jujur, dikarenakan ilmu pengetahuan merupakan jalan yang mempermudah seorang muslim untuk mendapatkan surga-Nya di akhirat kelak. Hal ini seperti perkataan Rasulullah di dalam sebuah hadis yang berbunyi,

وَمَنْ سَلَكَ طَرِيقًا يَلْتَمِسُ فِيهِ عِلْمًا سَهَّلَ اللَّهُ لَهُ بِهِ طَرِيقًا إِلَى الْجَنَّةِ

"Siapa yang menempuh jalan untuk mencari ilmu, maka Allah akan mudahkan baginya jalan menuju surga." (HR. Muslim, no. 2699).

Oleh karena itu, penulis ingin membuat suatu aplikasi yang bertujuan untuk mengecek tingkat bobot *similiarity* antara kode program satu dan lainnya.

Program ini dibuat dalam skala yang kecil dan ditujukan untuk bahasa pemrograman Java. Pada aplikasi ini dibuat menggunakan algoritma *levenshtein distance.* Sebelum penghitungan menggunakan algoritma, maka dimasukkan proses *prarocessing* yang berupa *case folding, tokenizing, stopword removal (filtering)* dan juga *sorting.* Dengan adanya tahapan *preprocessing* tersebut sebelum penghitungan algoritma, diharapkan hasil pengukuran tingkat *similiarity* bisa lebih akurat lagi.

1.2 Pernyataan Masalah

Dari latar belakang yang telah dipaparkan di atas, maka pernyataan masalah yang sesuai, yaitu berapa nilai *accuracy, precision, recall,* dan juga *F_1 Score* dari algoritma *levenshtein distance* pada kasus menghitung *similarity source code*.

1.3 Tujuan Penelitian

Mengukur *accuracy, precision, recall,* dan juga *F_1 Score* dari algoritma *levenshtein distance* pada kasus menghitung *similarity source code*.

1.4 Batasan Masalah

1. Menggunakan data *input* berupa *source code* bahasa pemrograman Java dalam satu *class file*.
2. Penentuan banyak jumlah *input* berdasarkan jumlah maksimal praktikan di laboratorium jurusan Teknik Informatika UIN Maulana Malik Ibrahim Malang.

1.5 Manfaat Penelitian

Penelitian ini akan menghasilkan *output* berupa persentase antara *source code* bahasa pemrograman Java. Sehingga, dari hal tersebut, aplikasi ini berguna untuk akademisi seperti praktikan maupun dosen pengampu praktikum untuk mengetahui apakah praktikan terindikasi melakukan plagiasi *source code* atau pun tidak, dan juga bisa dimanfaatkan secara umum yang membutuhkan pengecekan kesamaan antara *source code* bahasa pemrograman Java.

1.6 Sistematika Penulisan

Penelitian ini tersusun dalam laporan dengan terdiri dari beberapa bab pembahasan sebagai berikut :

Bab I Pendahuluan : Pada bab pertama ini berisi tentang latar belakang penelitian, identifikasi masalah, batasan masalah, tujuan penelitian, dan juga manfaat penelitian.

Bab II Tinjauan Pustaka : bab ini berisi tentang teori *levenshtein distance* dan juga penerapan perbandingan teks maupun *source code* dengan metode lainnya.

Bab III Analisis dan Perancangan : bab ini berisi analisis dan perancangan dalam pembuatan aplikasi menggunakan metode *levenshtein distance* yang meliputi *mockup, flowchart, pseudocode,* dan lain sebagainya.

Bab IV Uji Coba dan Pembahasan: bab ini berisi hasil implementasi algoritma *levenshtein distance* dan tahapan *preprocessing* pada aplikasi pendeteksian *similiarity* antara *source code*.

Bab V Penutup: bab ini berisi kesimpulan secara keseluruhan pada penelitian atau implementasi algoritma *levenshtein distance* dan tahap *preprocessing* pada aplikasi pencarian *similiarity source code.*

Daftar Pustaka: berisi daftar referensi yang tercantum dalam penelitian ini.

BAB II

STUDI PUSTAKA

2.1. Penelitian Terkait

Sutardi, *et al.* (2016) melakukan penelitian tentang aplikasi yang menggunakan algoritma *Levenshtein Distance* untuk membandingan tingkat kemiripan dari dua buah dokumen ataupun lebih. Pada prosesnya, penulis menggunakan tahapan *preprocessing*, yang terdiri dari *case folding*, *tokenizing*, *filtering, stemming* dan juga *sorting.* Penggunaan tahapan ini bertujuan untuk mendapatkan *keyword* yang akan digunakan sebagai pencocokan *string* atau perbandingan dokumen. Pada aplikasi ini, ada 3 (tiga) tipe *file* yang didukung, yaitu *doc, pdf,* dan juga *txt.* Pada pengujian pertama, penulis membandingan 10(sepuluh) dokumen asli dan 10(sepuluh) dokumen pembanding. Pada uji coba ini pula, terdapat dua perbedaan, yaitu yang menggunakan tahapan *stemming* dan yang tidak menggunakan. Hasil perbandingan dokumen-dokumen tersebut menunjukkan bahwa dokumen dengan kemungkinan kemiripan yang besar menghasilkan nilai *similiarity* di kategori atas, yaitu di atas 77% hingga 100%. Tahapan *preprocessing* yaitu *stemming* berpengaruh terhadap persentase tingkat kemiripan dan juga waktu prosesnya, walaupun tidak terlalu berpengaruh signifikan.

Pratama dan Pamungkas (2016) pada penelitian ini hampir serupa dengan penelitian sebelumnya, hanya saja perbedaannya di tiap pengetesan aplikasi, peneliti mencoba menggunakan tahapan *preprocessing* satu persatu, hingga

peneliti mengetahui dan didapatkan tahapan mana yang berpengaruh terhadap hasil akhir. Terdapat dua dataset dan satu data *real* yang diujicoba pada penelitian kali ini. Pada ujicoba dataset, peneliti mengubah struktur paragraf, yaitu mengubah kalimat aktif menjadi pasif, ataupun sebaliknya. Hal ini dimaksudkan untuk menguji tahapan *preprocessing*, yaitu *stopword removal*, *stemming*, dan *sorting* apakah mempengaruhi hasil akhir secara signifikan atau tidak. Selain itu dilakukan juga penghitungan waktu pemrosesan. Sedangkan pada pengujian data *real*, dokumen merupakan abstrak yang diambil secara acak dari internet tanpa ada pengubahan sama sekali. Hasil analisis pada data set pertama, yaitu penggunaan *stopword, stemming* dan *sorting* sekaligus didapatkan bobot *similiarity* tertinggi. Sedangkan pada ujicoba data set kedua, penggunaan *stopword* ataupun *stemming,* dapat menghasilkan bobot *similiarity* yang sama ketika digabungkan dengan proses *sorting.* Sedangkan pada ujicoba data *real*, hasil terbaik yang didapatkan adalah dari hasil *stemming* dan *sorting.* Hal ini diakibatkan karena banyaknya kata-kata umum pada dokumen yang tidak tereliminasi karena tidak adanya proses *stopword*.

Jimmy, dkk (2018) melakukan penelitian yang dimuat di Journal of Computer Science and Information System yang berfokus pada mengecek kemiripan *source code* dengan membandingkan dua buah metode uji, yaitu metode Fingerprint Based Distance, dan juga metode *Levenshtein Distance*. Bahasa pemrograman yang dipakai sebagai data *input* merupakan bahasa C++ sejjumlah 1225 pasangan *source code* yang didapatkan dari mahasiswa semester dua di Universitas Tarumanegara. Adapun tata cara untuk mendapatkan hasil

akhir yang berupa *accuracy, recall,* dan *precision* adalah dengan menggunakan beberapa uji coba dengan *input*-an data yang variatif, missal dengan membandingkan 5 buah hingga 50 *source code* yang berlaku tiap kelipatan lima. Hal ini dilakukan peneliti untuk mendapatkan hasil yang benar-benar terseleksi.

Pada penelitian ini, didapatkan hasil bahwa algoritma *Levenshtein Distance* lebih baik daripada algoritma Fingerprint Based Distance. Hal ini dibuktikan dengan hasil *precision* sebesar 98,33% dan juga *recall* sebesar 98,36% yang didapatkan pada uji coba di *threshold* 65%. Namun, para peneliti di akhir tulisannya juga menjelaskan kekurangan algoritma *Levenshtein Distance*, yaitu pada kasus peneliti melakukan perubahan pada struktur *source code* dan juga pada saat inisiasi variable dipindahkan tempatnya. Hal ini berpengaruh cukup signifikan terhadap hasil persentase akhir yang didapatkan walaupun data yang digunakan sama seperti sebelumnya.

Ahmad dan Eka (2014) pada penelitian kali ini mengusulkan metode deteksi plagiat dalam tiga tahapan berdasarkan penelitian dari Li Ping Zhang dan Dong Sheng Liu di Inner Mongolia Normal University. Terdapat tiga tahapan dalam prosesnya, yaitu *Code Formalization, Similiarity Calculation,* dan *Cluster Analysis*. Pada praktiknya, kedua peneliti menyesuaikan ketiga tahapan tersebut sesuai kondisi yang berlaku. Pada tahapan ketiga, di bagian *Indication Suspect*, peneliti memiliki tahapan-tahapan cabang lagi, yaitu *preprocessing, pooling, comparing,* dan *filtering.* Input dari penelitian ini adalah *source code* yang menggunakan bahasa pemrograman Java. Dataset yang digunakan sebanyak 13 potongan program yang diambil dari mahasiswa jurusan. Setelah dataset

diinputkan, maka ada dua tipe *output* utama, yaitu *Highlight Code Clone* dan Presentase Kemiripan. Semakin besar persentase kemiripan antar code tersebut, maka akan semakin besar pula indikasi bahwa *source code* tersebut hasil plagiat. Pada penelitian ini pula, peneliti membandingkan antara algoritma *Levenshtein Distance* dan Smith Waterman yang memiliki pendekatan dan tujuan yang sama, yaitu pencocokkan antar string.

Yulianingsih (2017) di penelitian ini melakukan pengetesan mencari data di *database* dengan menggunakan dua algoritma sekaligus. Namun, sebelum itu peneliti melakukan pengetesan satu per satu algoritma dan membandingkan keduanya, dan didapatkan bahwa algoritma tersebut memiliki keunggulan dan kelemahan masing-masing. Namun, jika menggabungkan keduanya, maka didapatkan hasil yang lebih baik. Khusus pada algoritma *Levenshtein Distance*, semakin kecil nilai skor atau bobot *similiarity* yang didapatkan sebelum pengetesan, maka bisa dipastikan bahwa kemiripannya semakin tinggi, dan hal ini pun berlaku sebaliknya.

Di pengujian pertama, peneliti mencoba menggunakan algoritma *Levenshtein Distance* untuk mencari kata “Thomas”. Jika dilihat dari hasil yang didapatkan, maka algoritma ini kurang efektif untuk pencarian kata di *database*. Berdasarkan tabel hasil, nama yang mengandung kata “Thomas” dan kombinasi kata “Thomas” pada urutan selanjutnya, yaitu didapatkan 10 buah. Sebagian dari hasil tersebut mendapatkan skor 10, yang berarti tingkat kemiripan dengan pencarian tidak terlalu mirip. Sedangkan hasil terbaik yang didapatkan yaitu skor poin 7.

Febriyanto, *et al.* (2006) di dalam penelitiannya memberikan contoh menggunakan elemen untuk menghitung kesamaan *source code* menggunakan algoritma Booyer-Moore. Contohnya ada 7 elemen dari 10 keseluruhan elemen dalam *array* objek. Setelah itu dimasukkan rumus untuk mengukur tingkat kesamaannya. Maka hasil yang didapatkan yaitu sebesar 70%. Namun, kekurangan dari penelitian ini, yaitu peneliti tidak menambahkan proses *preprocessing* yang berguna menyeleksi yang tidak perlu dimasukkan ketika penghitungan bobot *similiarity*.

Rizqa dan Wiguna (2014) menjelaskan bahwa algoritma Rabin-Karp, metode yang digunakan adalah *hash* untuk mencari string yang sama. Sebelum menghitung persamaannya, maka dicari terlebih dahulu nilai *hash* tersebut. Pada penelitian ini, nilai *hash* yang dihitung adalah 15 *substring*, dan hasil yang diperoleh memiliki kecocokan antara perhitungan *hash* secara manual maupun yang dihitung oleh sistem. *Input* dari aplikasi ini adalah berkas dari mahasiswa yang berjudul “tugasmodul.lisp”. Persentase kemiripan yang didapatkan yaitu tertinggi sebesar 100% dan terendah 93,02%. Hal ini menunjukkan bahwa berkas uji coba bisa dipastikan merupakan hasil *copy paste*.

2.2. *Levenshtein Distance*

Mawardi *et al.* (2017) menjelaskan bahwa *Levenshtein Distance* atau biasa juga dikenal dengan *Edit Distance*, adalah sebuah metrik pengukuran untuk menghitung perbedaan antara dua buah string ataupun lebih. *Levenshtein Distance* dari dua buah string adalah jumlah minimum titik-titik mutasi yang diperlukan untuk mengganti sebuah string dengan string lainnya. Ada tiga tahapan dari *Levenshtein Distance*, yaitu:

(i) *Subtitutions,* adalah sebuah operasi yang di mana berguna untuk mengganti sebuah karakter menjadi karakter lain. Contohnya kata "makan" menjadi "makar". Pada contoh sebelumnya, penulis mengganti kata "n" menjadi kata "r".

(ii) *Insertions,* adalah sebuah operasi yang di mana berguna untuk menyusupkan karakter baru pada sebuah kata, baik itu di bagian awal, tengah, maupun akhir kata tersebut. Contohnya yaitu kata "infomatika" jika disisipkan, maka menjadi kata "informatika", yaitu penambahan huruf "r" di bagian tengah kata awal.

(iii) *Deletions,* yaitu sebuah operasi yang digunakan untuk menghilangkan atau menghapus suatu karakter yang tidak diinginkan, bisa di bagian awal, tengah, maupun akhir sebuah string. Contohnya yaitu penulis ingin meghilangkan karakter "n" pada kata "Malang", yang menghasilkan kata "Malan".

Levenshtein Distance adalah hal yang dibutuhkan untuk mengkonversi sebuah string menjadi string lain. Di bawah ini adalah persamaan atau algoritma dari *Levenshtein Distance* yang digunakan untuk mengukur *similiarity* antara dua kata $a=a_{1....}a_n$ dan $b_{1...}b_m$ adalah:

$$d_{i0} = \Sigma_{k=1}^{i} w_{del}(b_k), \quad \text{for } 1 \le i \le m \tag{2.1}$$

$$d_{0j} = \Sigma_{k=1}^{j} w_{ins}(a_k), \quad \text{for } 1 \le j \le m \tag{2.2}$$

$$d_{ij} = \begin{cases} d_{i-1,j-1} & \text{for } 1\, a_j = b_j \\ \min \begin{cases} d_{i-1,j} + w_{del}(b_k) \\ d_{i-1,j} + w_{ins}(a_j) \\ d_{i-1,j-1} + w_{sub}(a_j, b_j) \end{cases} & \text{for } 1 \le i \le m, 1 \le j \le n \\ & \text{for } a_j \ne b_j \end{cases} \tag{2.3}$$

Keterangan:

a= Kata yang mewakili baris

b= Kata yang mewakili kolom

i= indeks huruf dari kata a

j= indeks huruf dari kata b

n= Panjang huruf dari kata a

m= Panjang huruf dari kata b

w= bobot nilai

Penghitungan algoritma *Levenshtein Distance* dimulai dari array dua dimensi di sudut kiri, dan nilai bobot dari algoritma ini didapatkan di matriks paling akhir yang berada di sudut kanan bawah.

2.3. *Text Processing*

Text Processing adalah sebuah tahapan yang dilakukan sebelum pengolahan data. Menurut tulisan dari Jaka Harjanta (2015) yang dipublikasikan di Jurnal Informatika Upgris yang membahas tentang *preprocessing* pada *data mining*, tahapan *preprocessing* sangat penting dilakukan terhadap aplikasi atau pun hal yang terkait data sebagai data ujinya. Hal ini dikarenakan jika tanpa adanya tahapan *preprocessing* ini, maka hasil akhir yang didapatkan mempunyai nilai yang tidak autentik, terlebih lagi pada data masukan yang memiliki kata-kata yang memengaruhi.

Dalam tulisannya, peneliti melakukan uji coba terhadap data uji teks dan melakukan beberapa tahapan di dalam *text processing*, seperti *stopword removal* atau *filtering* yang fungsi utamanya adalah menghilangkan kata-kata yang tidak diperlukan di dalam kasus penelitiannya, sehingga hasil akhir dari penelitiannya berupa kategorisasi menjadi tiga (3), yaitu aplikasi, bagus, dan buruk yang didapaatkan setelah memasukkan data di dalam matriks TF-IDF. Setelah peneliti mendapatkan hasil, Ia menjelaskan dalam beberapa konteks, *text preprocessing* juga sering terjadi kesalahan. Hal ini bisa terjadi diakibatkan banyak factor, salah satunya seperti kesamaan data masukan yang akan disaring maupun kata yang harusnya tidak termasuk. Hal tersebut bisa memengaruhi keluarannya. Oleh karena itu, peneliti mengusulkan untuk memperbanyak data masukan *preprocessing* lagi, dan juga untuk penelitian setelahnya bisa lebih jelih terhadap penelitian tersebut, karena *text preprocessing* bisa berbeda-beda sesuai penelitian yang akan dilakukan.

Selain dari pemaparan fungsi *text preprocessing* di paragraf sebelumnya, tahapan ini juga bertujuan untuk mempersiapkan dan melakukan penyeragaman data masukan sebelum di-*input*-kan ke dalam sebuah sistem ataupun aplikasi. *Preprocessing* menurut (Triawati, 2009) memiliki beberapa tahapan, yaitu:

1. *Case Folding*

   Menurut Triawati, *case folding* merupakan tahapan mengubah semua huruf menjadi huruf kecil. Karakter yang diterima yaitu ‘a’ hingga ‘z’.

2. *Tokenizing*

   Tokenizing merupakan tahap pemotongan kalimat atau string menjadi kata per kata.

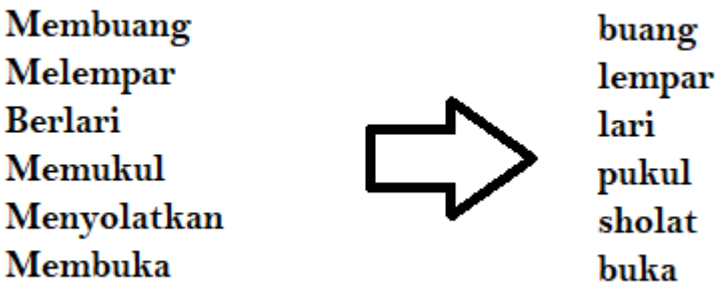
3. *Filtering*

   Filtering adalah proses setelah *tokenizing*. Proses ini bertujuan untuk mengambil kata-kata penting dari proses sebelumnya, yaitu *tokenizing*. Terdapat dua algoritma yang bisa digunakan pada proses *filtering* ini, yaitu *wordlist* (menyimpan kata-kata penting), dan algoritma *stoplist* (membuang kata yang kurang penting). *Stoplist/stopword* berisi kata-kata yang tidak terlalu penting, seperti ‘di’,’yang’,’dan’, dan lain-lainnya.

4. *Stemming*

   Stemming merupakan teknik *preprocessing* yang terdapat pada *Retrieval System* (RS) di mana kata-kata atau *string* diubah menjadi kata dasarnya. Penggunaan *stemming* sering digunakan pada kata yang menggunakan Bahasa Inggris sebagai bahasa utama. Hal ini dikarenakan untuk mencari akar kata (*root word*) cukup dengan menghilangkan sufiks, sedangkan jika diterapkan

pada dokumen yang menggunakan Bahasa Indonesia maka akan lebih rumit lagi.



Gambar 2.1 Contoh *Stemming*

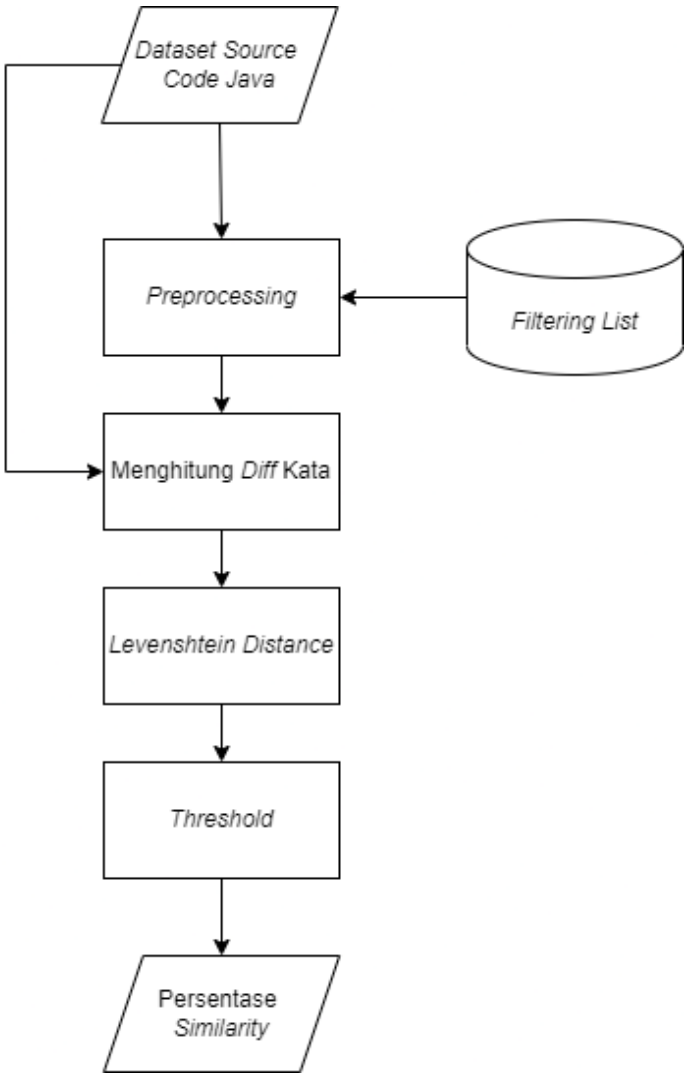
BAB III

METODE PENELITIAN

3.1. Prosedur Penelitian

Pada sub-bab ini membahas tentang penelitian yang akan dilakukan. Penelitian ini terdiri dari 6 bagian utama, seperti tertera di gambar di bawah. Studi pustaka bertujuan untuk memahami cara kerja algoritma *Levenshtein Distance* dan juga tahapan *preprocessing* apa saja yang ingin diterapkan pada pembuatan program ini. Setelah studi pustaka, berlanjut pada pembuatan aplikasi, yaitu mengaplikasikan yang didapatkan di tahapan sebelumnya. Kemudian mengumpulkan *input* yang dibutuhkan program, yaitu berupa *source code* yang menggunakan bahasa pemrograman Java. Setelah terkumpul dan diuji coba, hal yang perlu dilakukan adalah mengevaluasi sistem, apakah sudah berjalan sesuai rencana awal atau ada yang perlu diperbaiki lagi.

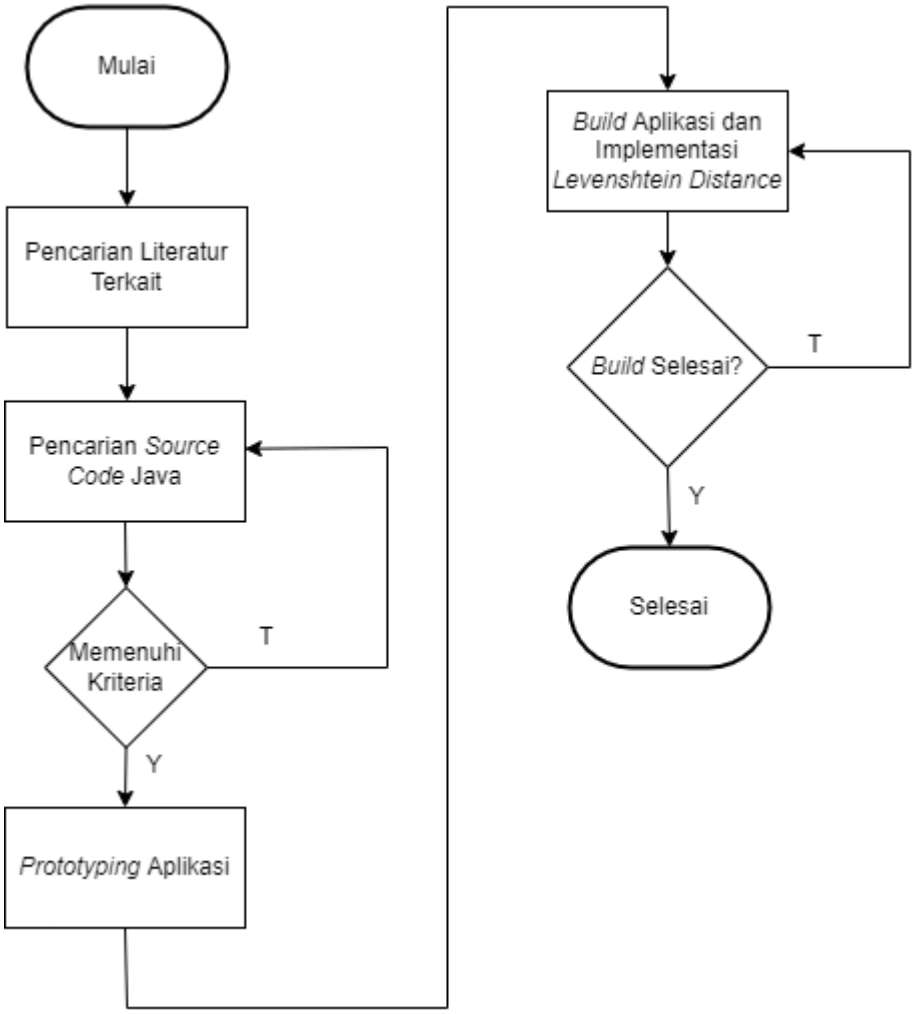
Pada penelitian pembuatan aplikasi untuk menghitung bobot *similiarity* dengan menggunakan algoritma *Levenshtein Distance* ini memiliki dua tahapan, yaitu pertama tahapan *preprocessing* dan yang terakhir yaitu tahapan intinya, yaitu *processing.* Tahapan *preprocessing* ini bertujuan untuk menyeleksi hal-hal yang dapat mengurangi akurasi pengukuran, seperti *token* dan lain sebagainya. Pada tahapan *preprocessing* ini terdapat 3 tahapan lagi, yaitu *case folding*, *tokenizing*, dan juga *stopword removal(filtering).*



Gambar 3.1 Sistem *Design*

Gambar di atas merupakan sistem *design* yang tiap komponennya berarti: 1) *Dataset source code* Java, merupakan *dataset* bahasa pemrograman Java yang telah dikumpulkan berdasarkan kriteria, seperti hanya per satu kelas tiap data. 2) *User* bisa memilih untuk menggunakan tahapan *preprocessing* ataupun tidak, hal ini berdasarkan empat (4) skenario uji coba yang akan digunakan. Adapun ketika melewati tahapan *preprocessing*, maka sistem akan mengambil data berupa kata-kata yang akan di-*filter* oleh sistem untuk dihilangkan. 3) Setelahnya, tiap *source code* Java dihitung jarak antar *string*-nya, yang kemudian hasilnya disebut *diff*. 4)

Setelahnya, nilai *diff* itu digunakan untuk tahapan *levehnshtein distance*, dan 5) menentukan *threshold* kategorisasi plagiat atau tidak plagiat, sehingga di bagan akhir nantinya, 6) menghasilkan *output* berupa persentase *similarity* antara *source code* yang telah di-*input*-kan di awal.



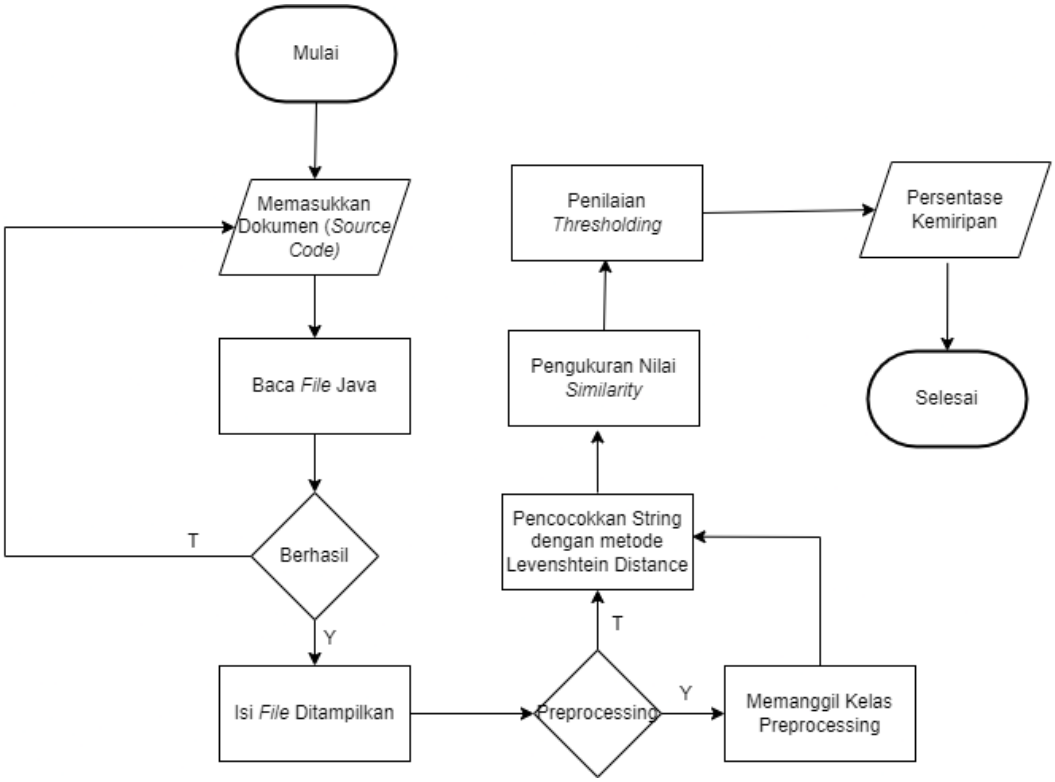
Gambar 3.2 Blok Diagram Penelitian

Adapun pada tahapan *processing*, terdapat beberapa tahap, yaitu menghitung bobot *similiarity*, kemudian dilanjutkan dengan memasukkan nilai

bobot ke rumus *Levenshtein Distance.* Sedangkan *output* dari sistem aplikasi ini, yaitu berupa persentase kesamaan antara *source code* dokumen satu dan lainnya.

Aplikasi pengecekan *similarity* antara *source code* ini yaitu sebuah aplikasi yang akan menghitung tingkat kesamaan antara *source code* utama dan *source code* pembanding. Pada aplikasi ini nantinya, akan ada tiga tahapan utama, yaitu *preprocessing*, setelah itu dilanjut pada pengecekan masukkan yang berupa *string*, dan yang terakhir mengukur *similarity*-nya dengan rumus yang sudah ada.

Adapun tahap *preprocessing* ditujukan untuk mengurangi hal-hal yang dapat menurunkan tingkat pengukuran, atau yang biasa disebut *noise* pada proses. Adapun proses selanjutnya, yaitu pengecekan masukkan yang berupa *string.* Pada tahap ini, hal yang dibutuhkan adalah *diff*. *Diff* merupakan suatu angka yang didapat setelah penghitungan menggunakan metode *Levenshtein Distance.* Nilai ini akan dibutuhkan untuk tahapan terakhir, yaitu pengukuran bobot *similarity*. Di tahap akhir ini, nilai *diff* yang sudah didapatkan, akan dimasukkan ke rumus, sehingga menghasilkan berapa persentase kesamaan yang didapatkan antara dua buah *source code* tersebut. Hal ini digambarkan seperti *flowchart* di bawah ini.



Gambar 3.3 *Flowchart* Program

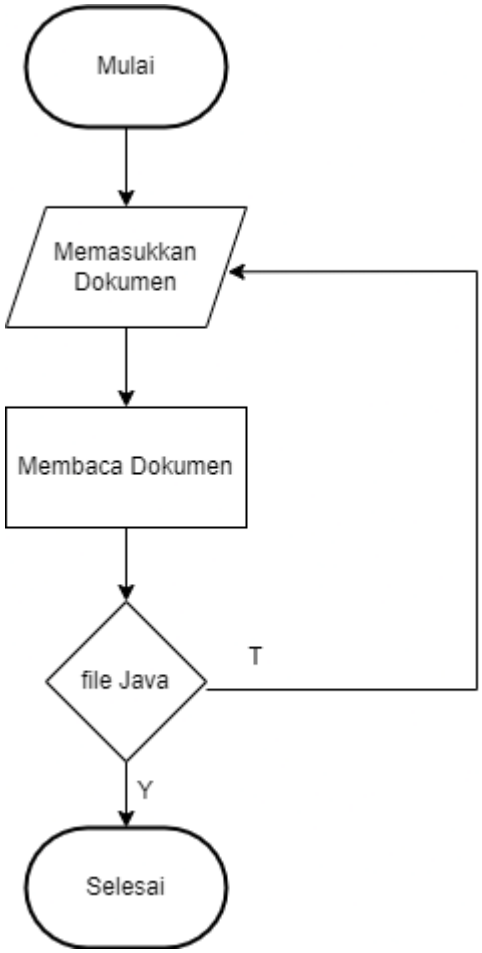
Setelah mengetahui *flowchart* dari aplikasi yang akan dibuat, maka diperlukan juga rambu-rambu yang digunakan pada saat proses pengerjaan *coding* yang berupa *pseudocode*. Hal ini agar memudahkan di dalam pembuatan program yang menggunakan metode *Levenshtein Distance.*

```
int LevenshteinDistance (char s[1...m], char t[1...n])
   declare int d[0...m, 0...n]
      for i from 0 to m
         d[i,0] := i
      for j from 0 to n
         d[0,j] := j
      for i from 1 to m
         for j from 1 to n
            if s[i] = t[j] then d[i,j] := d[i-1, j-1]
               else
            d[i,j] := minimum(
                d[i-1,j]+1, //penghapusan
                d[i,j-1]+1, //penyisipan
                d[i-1,j-1]+1 //penukaran
            )
      return d[m,n]
```

Gambar 3.4 *Pseudocode Levenshtein Distance*

3.2. Tahap *Preprocessing*

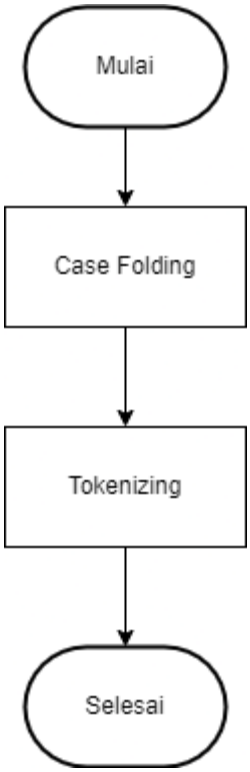
Pada bagian ini peneliti akan membahas lebih detail bagaimana proses *preprocessing* yang berguna untuk menyeleksi masukan sebelum diolah ke tahapan utama, yaitu proses *Levenshtein Distance.* Pada kasus penelitian ini, peneliti hanya menggunakan tiga (3) tahapan *preprocessing*, hal ini dikarenakan menyesuaikan dengan penelitian yang akan dilakukan. Pada dasarnya, *preprocessing* berdasarkan penjabaran salah satu penelitian di bab 2 memiliki 4 tahapan, yang satunya yaitu *stemming.* Namun, pada kasus ini, *stemming* tidak cocok digunakan karena tidak membutuhkan kalimat utama. Berikut ini adalah *flowchart* dari keseluruhan aplikasi, juga *flowchart* mandiri dari tahapan *preprocessing*.



Gambar 3.5 *Flowchart* Baca *File Source Code*

1. *Case Folding*

Case Folding adalah tahapan proses *preprocessing* yang di mana mengubah huruf-huruf pada dokumen yang dalam kasus ini adalah *source code* menjadi huruf kecil. Huruf yang berlaku atau bisa diubah yaitu hanya a-z. Namun, sebelum ke proses ini, ada tahapan *read file*, apakah *input* sudah berekstensi .java atau belum.



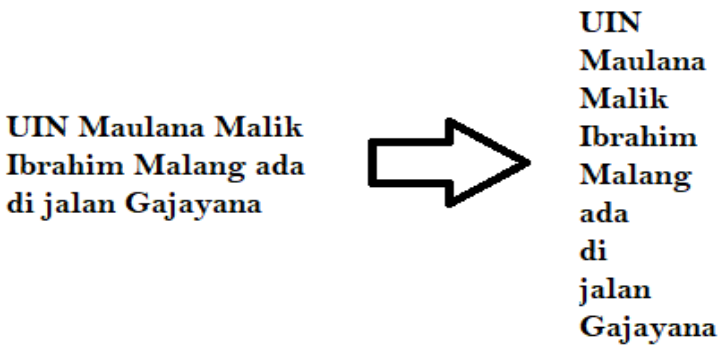
Gambar 3.6 *Flowchart preprocessing*

Setelah *file* dibaca sebagai berekstensi .java, maka proses selanjutnya akan berlanjut, yaitu tahapan *case folding*. Di bawah ini merupakan contoh penerapan *case folding.*

Gambar 3.7 Contoh Penerapan Proses *Case Folding*

2. *Tokenizing*

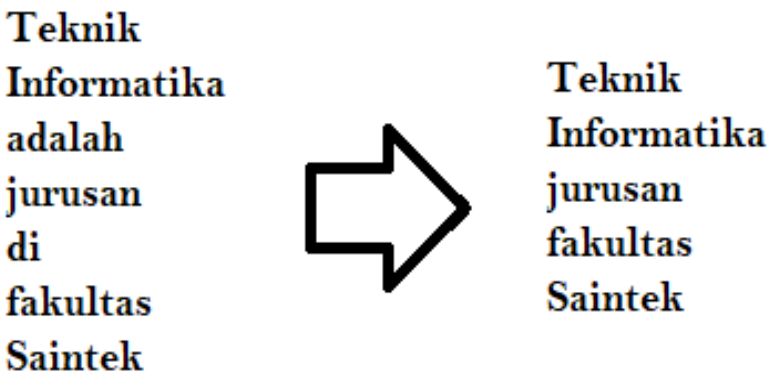
Tokenizing merupakan tahapan memecah kalimat menjadi kata per kata. Contohnya bisa dilihat seperti gambar di bawah ini.



Gambar 3.8 Contoh Penerapan Proses *Tokenizing*

3. Filtering

Filtering merupakan proses pengambilan kata-kata penting dari proses *tokenizing*.



Gambar 3.9 Contoh Penerapan Proses *Filtering*

3.3. Penghitungan *Levenshtein Distance*

Tahap ini merupakan proses penerapan metode *Levenshtein Distance* yang dilakukan setelah tahapan *preprocessing.* Setelah tahap *preprocessing,* kata-kata yang didapatkan dibandingkan antara satu dengan lainnya. Hal ini bertujuan untuk mencari nilai *diff* yang berguna untuk proses akhir, yaitu pada tahapan pengukuran *similarity*. Pada kasus ini, peneliti mencoba memberikan contoh sederhana penghitungan nilai *diff* dari dua buah kalimat. Hal ini dilakukan secara manual, agar diharapkan bisa lebih mengerti tentang konsep utama dari metode *Levenshtein Distance* tersebut.

Kalimat 1: jurusaninformatikafakultassaintek

Kalimat 2: jurusanmanajemenfakultasekonomi

Pada contoh kalimat di atas, akan dilakukan tiga (3) tahapan utama *Levenshtein Distance*, yaitu penyisipan, penghapusan, dan penukaran, tergantung kondisi apa yang tepat digunakan.

Tabel 3.1 Penghitungan Levenshtein Distance Manual

		j	u	r	u	s	a	n	i	n	f	o	r	m	a	t	i	k	a	f	a	k	u	l	t	a	s	s	a	i	n	t	e	k
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
j	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
u	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
r	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
u	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
s	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
a	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
n	7	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
m	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
a	9	8	7	6	5	4	3	2	2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
n	10	9	8	7	6	5	4	3	3	3	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
a	11	10	9	8	7	6	5	4	4	4	4	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
j	12	11	10	9	8	7	6	5	5	5	5	5	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
e	13	12	11	10	9	8	7	6	6	6	6	6	6	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
m	14	13	12	11	10	9	8	7	7	7	7	7	7	7	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
e	15	14	13	12	11	10	9	8	8	8	8	8	8	8	8	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
n	16	15	14	13	12	11	10	9	9	9	9	9	9	9	9	9	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
f	17	16	15	14	13	12	11	10	10	10	10	10	10	10	10	10	10	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
a	18	17	16	15	14	13	12	11	11	11	11	11	11	11	11	11	11	11	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
k	19	18	17	16	15	14	13	12	12	12	12	12	12	12	12	12	12	12	12	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
u	20	19	18	17	16	15	14	13	13	13	13	13	13	13	13	13	13	13	13	13	13	14	15	16	17	18	19	20	21	22	23	24	25	26
l	21	20	19	18	17	16	15	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	15	16	17	18	19	20	21	22	23	24	25	26
t	22	21	20	19	18	17	16	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	16	17	18	19	20	21	22	23	24	25	26
a	23	22	21	20	19	18	17	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	17	18	19	20	21	22	23	24	25	26
s	24	23	22	21	20	19	18	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	18	19	20	21	22	23	24	25	26
e	25	24	23	22	21	20	19	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	19	20	21	22	23	24	25	26
k	26	25	24	23	22	21	20	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	20	21	22	23	24	25	26
o	27	26	25	24	23	22	21	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	21	22	23	24	25	26
n	28	27	26	25	24	23	22	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	22	23	24	25	26
o	29	28	27	26	25	24	23	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	23	24	25	26
m	30	29	28	27	26	25	24	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	24	25	26
i	31	30	29	28	27	26	25	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	25	26

Dari matriks pengukuran di atas, didapatkan bahwa nilai *diff* yang didapatkan yaitu 26. Nilai *diff* merupakan nilai akhir dari pengukuran *Levenshtein Distance* yang letaknya berada di pojok kanan bawah.

3.2.3 Penghitungan Nilai *Similarity*

Setelah sebelumnya melakukan pengukuran nilai *distance* untuk mencari nilai *diff*, maka langkah selanjutnya mengukur nilai *similarity*.

Nilai *diff* = 26

Max (CS, ST) = 33

Rumus Nilai *Similarity* $= (1 - \frac{diff}{Max\ (CS,ST)}) * 100\%$

$= (1 - \frac{26}{33}) * 100\%$

$= (\frac{33}{33} - \frac{26}{33}) * 100\%$

$= 0{,}212 * 100\%$

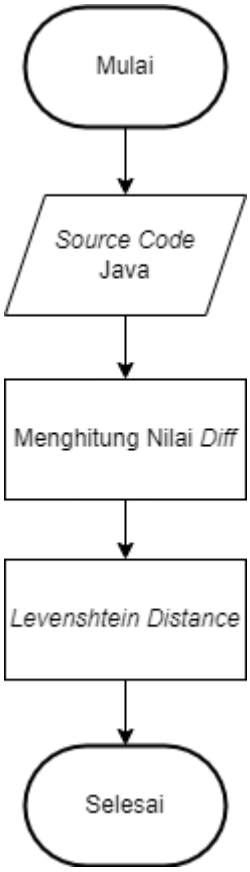
$= 21{,}2\ \%$

Jadi, nilai kesamaan / *similarity* yang didapatkan dari contoh dua buah kalimat di atas adalah sebesar 21,2%

3.3. Rancangan Percobaan

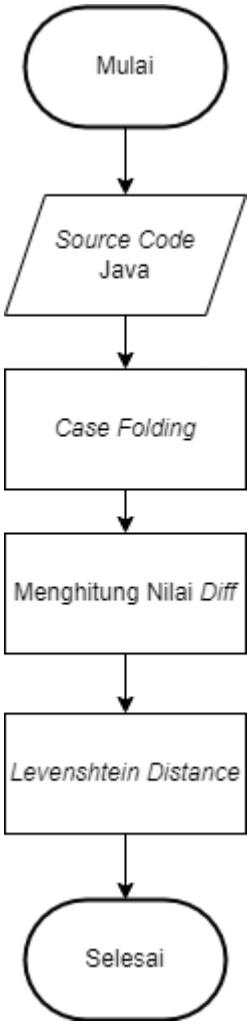
Di sub-bab ini membahas bagaimana penelitian akan dilakukan. Dari pemaparan sebelumnya terkait algoritma *Levenshtein Distance* dan tahapan *preprocessing*, maka penelitian ini akan menggabungkan keseluruhan kemungkinan yang terjadi, yang nantinya didapatkan ada 4 skenario yang akan diimplementasikan, yaitu:

1. Skenario Pertama, yaitu percobaan dengan menggunakan algoritma *Levenshtein Distance* autentik, tanpa menggabungkan apa pun. Hal ini dilakukan agar melihat seberapa efektif algoritma ini untuk mengukur tingkat *similarity* secara alami antara *source code* satu dan lainnya.



Gambar 3.10 *Flowchart* Skenario Ke-1

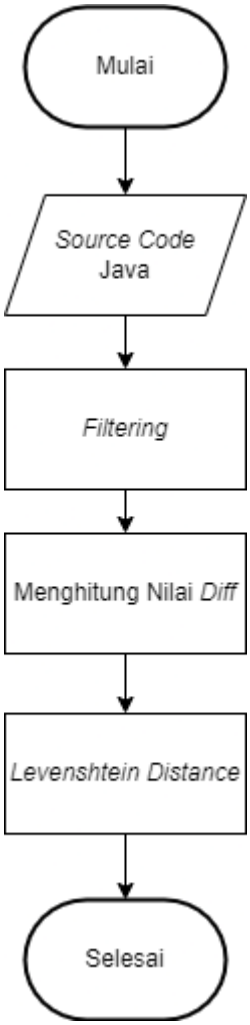
2. Skenario Kedua, yaitu percobaan dengan menggunakan algoritma *Levenshtein Distance* ditambah tahapan *preprocessing* berupa *Case Folding*, yaitu tahapan di mana setelah algoritma *Levenshtein Distance* dihitung, maka akan dilakukan pengecilan seluruh huruf. Hal ini dilakukan agar perhitungan bisa lebih maksimal, dikarenakan bahasa Java yang *case-sensitive*.



Gambar 3.11 *Flowchart* Skenario Ke-2

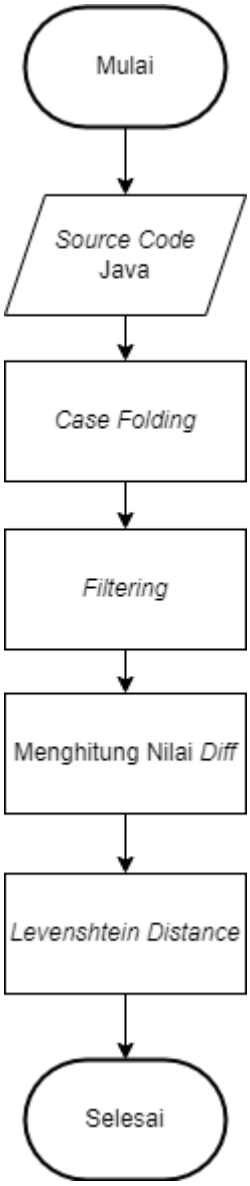
3. Skenario Ketiga, yaitu percobaaan dengan menggunakan algoritma *Levenshtein Distance* ditambah tahapan *preprocessing* berupa *Filtering*, yaitu tahapan di mana setelah algoritma *Levenshtein Distance* dihitung, maka akan dilakukan

tahapan penghapusan kata atau karakter yang tidak diperlukan, di mana dalam hal ini berupa kata-kata wajib di bahasa pemrograman Java, seperti *Public*, dan lain sebagainya. Hal ini dilakukan agar bisa mengurangi terlalu tingginya persentase akhir yang didapatkan.



Gambar 3.12 *Flowchart* Skenario Ke-3

4. Skenario Keempat, yaitu percobaan dengan menggunakan algoritma *Levenshtein Distance* ditambah kedua tahap *preprocessing* pada skenario kedua dan ketiga. Hal ini agar melihat hasil persentase akhir jika tahapan *preprocessing* dipakai secara maksimal.



Gambar 3.13 *Flowchart* Skenario Ke-4

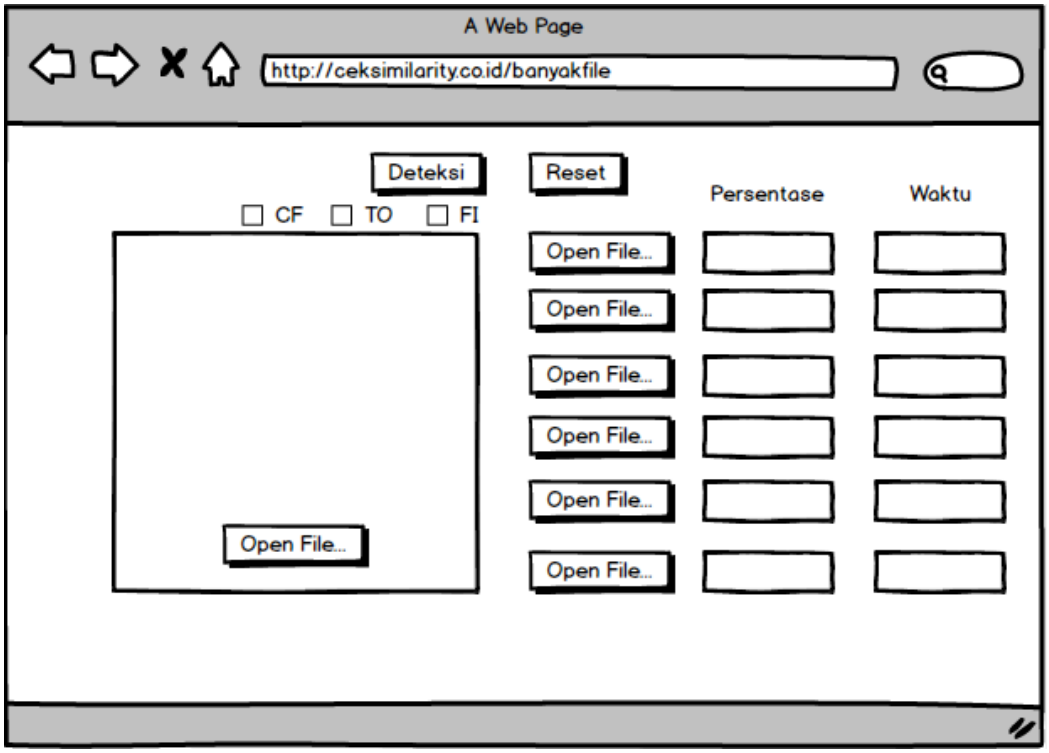
3.4. Desain Tampilan

Pengujian aplikasi *similarity checker* ini direncanakan akan dibangun menggunakan dua tampilan atau *layer user interface. Layout* yang pertama yaitu beranda sekaligus untuk memasukkan *input* dokumen acuan dan dokumen pembanding. Setelah memilih dokumen, maka *user* memilih proses *preprocessing*

apa saja yang ingin ditambahkan pada pengukuran aplikasi ini. Ada tiga (3) tahapan *preprocessing*, yaitu *case folding*, *tokenizing,* dan *filtering.* Hal ini dimaksudkan agar *user* bisa membandingkan persentase jika menggunakan tahapan-tahapan tersebut maupun pada saat tidak menggunakannya.

Pada bagian ini, merupakan pengecekan *similarity* untuk banyak dokumen/*source code*. Namun, pada kasus ini yang ditampilkan teks/*source code*-nya hanyalah *input*-an awal, sedangkan *input*-an lain hanya diperlihatkan persentase dan total waktu yang dibutuhkan untuk mendapatkan hasil akhir.

Di tahapan ini, nantinya tiap *source code* dibandingkan dengan *source code* lainnya, sehingga ada sekitar 200 hasil penghitungan algoritma *Levenshtein Distance*. Hal ini dilakukan agar dosen mengetahui tingkat kemiripan tiap siswa tanpa harus meng-*input*kan data *source code* secara berulang kali, cukup sekali input sebanyak 20 data yang di mana mewakili banyaknya jumlah mahasiswa dalam suatu kelas, kemudian akan muncul persentase dari tiap *source code* tersebut.



Gambar 3.14 Halaman Deteksi Banyak *File*

BAB IV

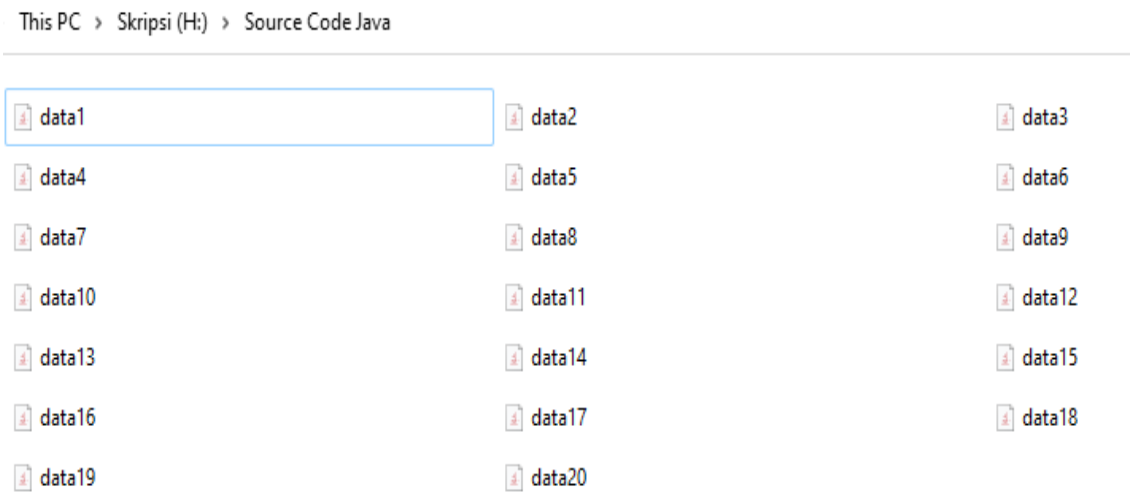
UJI COBA DAN PEMBAHASAN

Pada bab IV ini berisi perihal skenario pengujian sistem, serta pembahasan hasil dari uji coba beberapa skenario tersebut, dan juga diakhiri bagaimana penelitian dan sistem yang telah dibuat berintegrasi dengan Islam. Uji coba dan pembahasan dilakukan untuk mengukur tingkat akurasi dan lainnya dari algoritma *Levenshtein Distance* pada kasus pengecekan *similarity code* bahasa pemrograman Java.

4.1 Skenario Uji Coba

Pada sub-bab ini menjelaskan tentang pelaksanaan beberapa skenario uji coba yang diolah dari data uji yang telah peneliti kumpulkan. Terdapat beberapa uji coba dengan menggunakan data yang sama, hal ini dilakukan untuk mengukur fungsionalitas sistem menjalankan algoritma *Levenshtein Distance* yang digunakan pada kasus mengukur kesamaan *source code*.

Ada pun data uji yang digunakan oleh peneliti pada penelitian kali ini berjumlah 20 data yang berupa *source code* bahasa pemrograman Java mendasar seperti tugas-tugas praktikum mahasiswa pada umumnya. Data di bawah ini merupakan data uji yang dikumpulkan



Gambar 4.1 Data Uji Keseluruhan

Berikut di bawah merupakan isi dari beberapa contoh data uji yang digunakan

```
public class MyClass {
  public static void main(String[] args) {
    int myNum = 5;               // integer (whole number)
    float myFloatNum = 5.99f;    // floating point number
    char myLetter = 'D';         // character
    boolean myBool = true;       // boolean
    String myText = "Hello";     // String
    System.out.println(myNum);
    System.out.println(myFloatNum);
    System.out.println(myLetter);
    System.out.println(myBool);
    System.out.println(myText);
  }
}
```

Gambar 4.2 Isi data1.java

```
public class bacaFile {
    public static void main(String[] args) {
        FileInputStream input = null ;
        int data ;
        // Membuka File
        try
        {
            input = new FileInputStream("G:/sister/bacalah.txt") ;

        }catch(FileNotFoundException e)
        {
            System.out.println("File gak nemu");
        }

        try
        {
            while((data = input.read())!= -1)
            {
                System.out.print((char)data);
            }
            System.out.println();
        }
        catch(IOException e){
            System.out.println(e.getMessage());
        }

        try{
            input.close();

        }
        catch(IOException E)
        {

        }
    }
}
```

Gambar 4.3 Isi data20.java

Dari contoh kedua gambar yang menampilkan *source code* bahasa pemrograman Java di atas, terlihat juga bahwa batasan masalah terhadap penelitian ini yaitu pada *source code* yang masih dalam satu kelas dan sederhana, seperti pada praktikum yang dilakukan praktikan.

Pada penelitian ini, peneliti melakukan beberapa skenario uji terhadap data-data uji di atas, yaitu:

1. Skenario Pertama

   Skenario pertama merupakan uji coba yang dilakukan tanpa tahap *preprocessing*, yaitu dengan algoritma *Levenshtein Distance* autentik.

2. Skenario Kedua

   Skenario kedua merupakan uji coba yang dilakukan dengan skema *Levenshtein Distance* ditambah tahapan *preprocessing case folding*.

3. Skenario Ketiga

   Skenario ketiga merupakan uji coba yang dilakukan dengan skema *Levenshtein Distance* ditambah tahapan *preprocessing filtering*.

4. Skenario Keempat

   Skenario keempat merupakan uji coba yang dilakukan dengan skema *Levenshtein Distance* ditambah tahapan keseluruhan *preprocessing*, yaitu *case folding* dan *filtering*.

4.2 Hasil Uji Coba

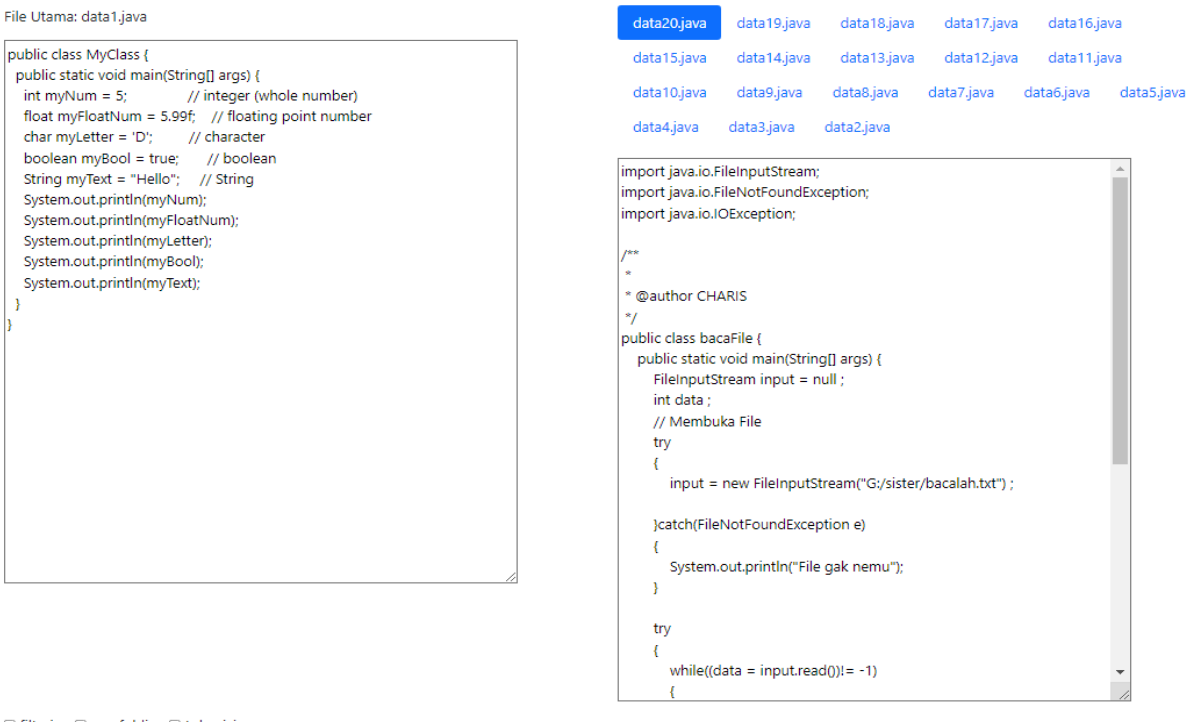
Di sub-bab kali ini, peneliti akan membahas hasil setelah pengujian yang dilakukan pada aplikasi deteksi *similarity* pada *source code* bahasa pemrograman Java yang berbasis *web*. Aplikasi ini dibuat dengan menerapkan algoritma *Levenshtein Distance* sebagai acuan utama untuk mengukur jarak antar *string*, yang di mana pada kasus kali ini berupa *source code*. Ada pun sebagai pembanding untuk mengukur tingkat efektivitas dari algoritma *Levenshtein Distance* ini, maka peneliti menggunakan fungsi dasar yang ada di bahasa pemrograman *Hypertext Preprocessor* (PHP), yaitu fungi similar_text. Fungsi ini serupa dengan *Levenshtein Distance* yang mengukur jarak string, namun dengan

rumus yang berbeda. Hal ini yang mendasari peneliti mengambil fungsi ini sebagai pembanding.

Pada pengujian ini, peneliti meng-*input* satu buah *file source code* yang dijadikan acuan sebagai pembanding. Setelahnya, di-*input*-kan lagi sembilan belas (19) *file source code* pemrograman Java yang berbeda sebagai alat ukur untuk dibandingkan dengan *file* pertama tadi. Acuan mengambil total dua puluh (20) *file*, yaitu dari jumlah rata-rata praktikan di lab pemrograman kampus UIN Maulana Malik Ibrahim Malang, khususnya di jurusan Teknik Informatika. Ada pun ambang batas (*threshold*) plagiasi yang digunakan adalah maksimal 25%, lebih dari itu maka terdeteksi sebagai plagiat. Angka tersebut didapat dari acuan standarisasi kampus terhadap plagiasi skripsi dan juga tugas lainnya.

Di pengujian kali ini pula, selain membandingkan dengan algoritma *Levenshtein Distance* murni, peneliti juga menambahkan tahapan *preprocessing* yang digunakan agar bisa memilah kata-kata yang bisa jadi penghambat ketika menggunakan algoritma *Levenshtein Distance* secara murni.

Setelah data dimasukkan, maka akan muncul tampilan seperti di bawah ini



Gambar 4.4 Tampilan Aplikasi Setelah Data Di-*input*

Menurut Kamber dan Han (2011) di tulisannya, *confusion matrix* didefinisikan sebagai alat untuk menganalisa apakah suatu *classifier* atau pun algoritma tersebut baik ataukah masih di bawah rata-rata pada kasus tertentu. Ada empat klasifikasi pada *confusion matrix,* yaitu True Positive, False Positive, False Negative, dan juga True Negative. Adapun setelah ini, keempat klasifikasi tersebut disingkat secara berurut masing-masing menjadi TP, FP, FN, dan juga TN.

Tabel 4.1 Pendefinisian *Confusion Matrix*

No	Variabel	Ketentuan
1	True Positive (TP)	Kasus di mana *predicted* (*Levenshtein Distance*) dan juga *Ground Truth* terdeteksi plagiat
2	False Positive (FP)	Kasus di mana *predicted* (*Levenshtein Distance*) terdeteksi plagiat, sedangkan *Ground Truth* terdeteksi tidak plagiat
3	False Negative (FN)	Kasus di mana *predicted* (*Levenshtein Distance*) terdeteksi tidak plagiat, sedangan *Ground Truth* terdeteksi plagiat
4	True Negative (TN)	Kasus di mana *predicted* (*Levenshtein Distance*) dan juga *Ground Truth* terdeteksi tidak plagiat

Setelahnya, data-data dari *confusion matrix* tersebut akan dimasukkan untuk menghitung *accuracy, recall, precision*, dan juga F_1 *score*. Adapun definisi singkat dan rumusnya seperti di bawah ini

- ***Recall*** atau *sensitivity*: menggambarkan keberhasilan model dalam menemukan kembali sebuah informasi.

$$\mathbf{TP / (TP + FN)} \tag{4.1}$$

- ***Precision*** menggambarkan akurasi antara data yang diminta dengan hasil prediksi yang diberikan oleh model.

$$\mathbf{TP / TP + FP} \tag{4.2}$$

- ***Accuracy*** menggambarkan seberapa akurat model dalam mengklasifikan dengan benar.

(TP+TN) /(TP + FP + FN + TN) (4.3)

- ***F-1 Score*** menggambarkan perbandingan rata-rata precision dan recall yang dibobotkan. Accuracy tepat kita gunakan sebagai acuan performansi algoritma jika dataset kita memiliki jumlah data False Negatif dan False Positif yang sangat mendekati (symmetric). Namun jika jumlahnya tidak mendekati, maka sebaiknya kita menggunakan F1 Score sebagai acuan.

(2 * Recall * Precision) / (Recall + Precision) (4.4)

Dari dua puluh (20) data uji yang digunakan, data-data itu kemudian saling dihitung persentase antara data uji. Adapun skenario yang digunakan sebanyak empat variasi pnghitungan berbeda, yang setelah disebut skenario pertama, kedua, ketiga, dan keempat. Jadi, perhitungan per skenario sebanyak 380 perhitungan, sehingga total keseluruhan 1520 perhitungan oleh sistem.

Namun, perhitungan TP, FP, FN, dan TN dari tabel-tabel di bawah ini, hasilnya dibagi dua, hal itu dikarenakan nilai tersebut sudah didapatkan, sehingga terhitung sebagai perulangan tabel.

Tabel 4.2 *Output* Persentase *Predicted* Skenario Pertama

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	100	23.11	34.35	29.28	38.25	36.85	20.51	24.77	37.05	34.46	30.63	26.89	28.49	28.35	29.08	28.29	30.47	31.28	16.17	29.82
2	23.11	100	17.31	25.91	20.51	23.00	8.04	10.51	21.57	31.31	18.22	24.15	22.71	21.26	24.32	22.54	17.07	16.28	7.05	12.62
3	34.35	17.31	100	38.18	37.65	32.76	29.09	36.72	34.74	28.27	34.08	37.91	38.44	41.22	40.82	35.80	31.18	27.95	24.02	35.56
4	29.28	25.91	38.18	100	34.10	29.46	16.84	22.10	31.37	28.13	34.00	50.80	48.13	53.54	53.64	46.65	30.89	27.95	13.89	25.96
5	38.25	20.51	37.65	34.10	100	48.62	19.82	25.34	44.47	36.87	31.24	22.10	28.96	28.94	26.59	20.54	33.15	35.13	18.04	34.20
6	36.85	23.00	32.76	29.46	48.62	100	16.68	21.53	35.78	34.88	30.78	29.16	33.75	33.66	30.00	29.02	33.15	28.21	14.60	26.90
7	20.51	8.04	29.09	16.84	19.82	16.68	100	48.27	18.22	13.85	22.86	18.70	19.39	20.35	19.71	18.70	22.32	23.49	26.81	32.61
8	24.77	10.51	36.72	22.10	25.34	21.53	48.27	100	24.19	17.78	27.79	23.90	25.41	26.78	28.29	23.47	27.65	27.79	29.35	33.33
9	37.05	21.57	34.74	31.37	44.47	35.78	18.22	24.19	100	36.03	32.47	23.92	34.58	32.09	30.45	27.01	33.29	32.18	16.78	29.61
10	34.46	31.31	28.27	28.13	36.87	34.88	13.85	17.78	36.03	100.00	28.48	29.38	32.08	30.51	29.32	30.58	25.81	27.95	12.87	22.31
11	30.63	18.22	34.08	34.00	31.24	30.78	22.86	27.79	32.47	28.48	100.00	38.59	44.10	40.74	36.29	34.46	25.53	27.95	18.09	28.88
12	26.89	24.15	37.91	50.80	22.10	29.16	18.70	23.90	23.92	29.38	38.59	100.00	48.96	71.46	56.36	46.65	27.93	26.54	13.84	24.19
13	28.49	22.71	38.44	48.13	28.96	33.75	19.39	25.41	34.58	32.08	44.10	48.96	100	50.39	41.25	46.25	32.02	30.90	15.91	27.01
14	28.35	21.26	41.22	53.54	28.94	33.66	20.35	26.78	32.09	30.51	40.74	71.46	50.39	100	56.30	44.09	30.18	30.26	15.81	26.69
15	29.08	24.32	40.82	53.64	26.59	30.00	19.71	28.29	30.45	29.32	36.29	56.36	41.25	56.30	100	48.44	31.73	28.46	15.10	28.26
16	28.29	22.54	35.80	46.65	20.54	29.02	18.70	23.47	27.01	30.58	34.46	46.65	46.25	44.09	48.44	100	26.66	27.05	14.55	26.49
17	30.47	17.07	31.18	30.89	33.15	33.15	22.32	27.65	33.29	25.81	25.53	27.93	32.02	30.18	31.73	26.66	100	25.64	21.95	32.12
18	31.28	16.28	27.95	27.95	35.13	28.21	23.49	27.79	32.18	27.95	27.95	26.54	30.90	30.26	28.46	27.05	25.64	100	21.08	31.49
19	16.17	7.05	24.02	13.89	18.04	14.60	26.81	29.35	16.78	12.87	18.09	13.84	15.91	15.81	15.10	14.55	21.95	21.08	100	29.30
20	29.82	12.62	35.56	25.96	34.20	26.90	32.61	33.33	29.61	22.31	28.88	24.19	27.01	26.69	28.26	26.49	32.12	31.49	29.30	100

Tabel di atas berisi tentang nilai perbandingan antara *source code* data1.java yang dibandingkan dengan data2.java hingga data20.java. Data tersebut tanpa disaring terlebih dahulu menggunakan tahapan *preprocessing*, yang hanya menggunakan algoritma *Levenshtein Distance* secara autentik. Di tabel terdapat 400 hasil perhitungan yang didapatkan.

Tabel 4.3 *Output* Persentase *Ground Truth* Skenario Pertama

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	100	28.16	34.79	33.22	33.97	41.62	20.77	26.23	45.93	24.31	30.30	36.98	40.73	37.03	31.63	25.47	26.09	32.92	18.99	34.50
2	28.16	100	19.70	33.70	18.04	23.69	9.59	12.44	28.91	39.15	25.71	29.07	23.99	27.34	29.67	28.03	21.65	26.89	8.98	16.06
3	34.79	19.70	100	46.77	46.01	33.04	27.49	27.12	31.93	22.84	34.04	37.46	43.65	38.26	54.30	34.02	37.11	37.35	23.30	36.01
4	33.22	33.70	46.77	100	27.99	42.36	21.38	22.08	43.55	34.01	47.04	56.14	57.93	62.51	66.83	46.59	40.82	30.73	21.01	35.66
5	33.97	18.04	46.01	27.99	100	31.67	23.45	14.37	45.61	22.28	35.88	23.83	42.45	23.99	31.81	32.65	42.87	30.31	24.93	29.58
6	41.62	23.69	33.04	42.36	31.67	100	23.23	23.42	39.25	25.70	35.19	44.55	30.68	48.72	32.65	29.94	38.50	31.19	16.19	36.40
7	20.77	9.59	27.49	21.38	23.45	23.23	100	50.64	24.51	12.60	19.68	20.90	20.53	20.13	26.15	22.02	27.38	19.27	16.57	28.91
8	26.23	12.44	27.12	22.08	14.37	23.42	50.64	100	33.06	16.41	22.14	26.81	22.26	24.67	40.79	27.98	27.26	20.01	14.57	31.69
9	45.93	28.91	31.93	43.55	45.61	39.25	24.51	33.06	100	33.65	35.82	34.24	39.64	37.77	36.32	40.19	35.45	37.54	24.11	35.70
10	24.31	39.15	22.84	34.01	22.28	25.70	12.60	16.41	33.65	100	24.64	29.95	27.94	25.57	27.83	28.83	22.93	33.18	9.73	21.27
11	30.30	25.71	34.04	47.04	35.88	35.19	19.68	22.14	35.82	24.64	100	47.99	48.01	44.79	42.82	41.42	32.45	38.52	23.38	34.12
12	36.98	29.07	37.46	56.14	23.83	44.55	20.90	26.81	34.24	29.95	47.99	100	55.71	66.10	70.08	40.81	33.80	28.55	14.01	33.76
13	40.73	23.99	43.65	57.93	42.45	30.68	20.53	22.26	39.64	27.94	48.01	55.71	100	62.15	53.70	56.25	25.23	35.56	19.24	31.55
14	37.03	27.34	38.26	62.51	23.99	48.72	20.13	24.67	37.77	25.57	44.79	66.10	62.15	100	67.93	39.54	35.00	31.68	20.23	35.31
15	31.63	29.67	54.30	66.83	31.81	32.65	26.15	40.79	36.32	27.83	42.82	70.08	53.70	67.93	100	54.95	24.37	33.44	21.38	33.60
16	25.47	28.03	34.02	46.59	32.65	29.94	22.02	27.98	40.19	28.83	41.42	40.81	56.25	39.54	54.95	100	29.73	31.43	17.35	28.00
17	26.09	21.65	37.11	40.82	42.87	38.50	27.38	27.26	35.45	22.93	32.45	33.80	25.23	35.00	24.37	29.73	100	27.27	16.33	36.57
18	32.92	26.89	37.35	30.73	30.31	31.19	19.27	20.01	37.54	33.18	38.52	28.55	35.56	31.68	33.44	31.43	27.27	100	19.18	27.49
19	18.99	8.98	23.30	21.01	24.93	16.19	16.57	14.57	24.11	9.73	23.38	14.01	19.24	20.23	21.38	17.35	16.33	19.18	100	30.70
20	34.50	16.06	36.01	35.66	29.58	36.40	28.91	31.69	35.70	21.27	34.12	33.76	31.55	35.31	33.60	28.00	36.57	27.49	30.70	100

Tabel di atas berisi tentang nilai perbandingan antara *source code* data1.java yang dibandingkan dengan data2.java hingga data20.java. Data tersebut tanpa disaring terlebih dahulu menggunakan tahapan *preprocessing*, yang hanya tahapan *Ground Truth* autentik. Di tabel terdapat 400 hasil perhitungan yang didapatkan.

Tabel 4.4 *Confusion Matrix* Skenario Pertama

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	TP	FN	TP	TP	TP	TP	TN	FN	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
2	FN	TP	TN	TP	TN	TN	TN	TN	FN	TP	FN	FN	TN	FN	FN	FN	TN	FN	TN	TN
3	TP	TN	TP	TP	TP	TP	TP	TP	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
4	TP	TP	TP	TP	TP	TP	TN	TN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
5	TP	TN	TP	TP	TP	TP	TN	FP	TP	FP	TP	TN	TP	FP	TP	FN	TP	TP	TN	TP
6	TP	TN	TP	TP	TP	TP	TN	TN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
7	TN	TN	TP	TN	TN	TN	TP	TP	TN	TN	TN	TN	TN	TN	FN	TN	FN	TN	FP	TP
8	FN	TN	TP	TN	FP	TN	TP	TP	FN	TN	FP	FN	FP	FP	TP	FN	TP	FP	FP	TP
9	TP	FN	TP	TP	TP	TP	TN	FN	TP	TP	TP	FN	TP	TP	TP	TP	TP	TP	TN	TP
10	FP	TP	FP	TP	FP	TP	TN	TN	TP	TP	FP	TP	TP	TP	TP	TP	FP	TP	TN	TN
11	TP	FN	TP	TP	TP	TP	TN	FP	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
12	TP	FN	TP	TP	TN	TP	TN	FN	FN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	FN
13	TP	TN	TP	TP	TP	TP	TN	FP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
14	TP	FN	TP	TP	FP	TP	TN	FP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
15	TP	FN	TP	TP	TP	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	TP	FP	TP	TN	TP
16	TP	FN	TP	TP	FN	TP	TN	FN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
17	TP	TN	TP	TP	TP	TP	FN	TP	TP	FP	TP	TP	TP	TP	FP	TP	TP	TP	TN	TP
18	TP	FN	TP	TP	TP	TP	TN	FP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
19	TN	TN	TN	TN	TN	TN	FP	FP	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	TP	TP
20	TP	TN	TP	TP	TP	TP	TP	TP	TP	TN	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP

TP	FP	FN	TN
118	14	17	41

Tabel di atas merupakan perhitungan *confusion matrix* yang didapatkan dari perhitungan tabel 4.2 dan tabel 4.3.

Tabel 4.5 *Output* Persentase *Predicted* Skenario Kedua

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	100	23.11	34.61	29.68	38.45	37.85	20.88	25.20	37.45	34.86	31.39	27.69	29.08	29.13	30.08	28.88	31.03	31.92	16.73	30.55
2	23.11	100	17.31	26.46	20.74	23.26	8.31	10.51	21.81	31.31	18.99	24.37	23.13	21.46	24.55	22.99	17.63	16.41	7.45	13.14
3	34.61	17.31	100	38.44	37.91	32.76	29.36	37.22	35.14	28.53	34.61	38.04	38.71	41.35	41.61	36.20	32.36	28.85	24.43	35.77
4	29.68	26.46	38.44	100	34.56	29.72	17.05	22.53	31.62	28.97	34.76	50.80	48.33	53.74	54.32	47.10	31.88	28.08	14.14	26.59
5	38.45	20.74	37.91	34.56	100	48.62	19.87	25.34	44.47	37.33	31.24	22.32	29.17	28.94	27.05	20.76	33.29	35.26	18.30	34.41
6	37.85	23.26	32.76	29.72	48.62	100	16.78	21.53	35.78	35.66	30.78	29.38	34.17	33.86	30.00	29.02	33.57	28.33	14.85	27.01
7	20.88	8.31	29.36	17.05	19.87	16.78	100	48.53	18.38	14.06	23.18	18.97	19.77	20.51	19.82	19.13	22.75	24.03	27.37	32.82
8	25.20	10.51	37.22	22.53	25.34	21.53	48.53	100.00	24.26	18.07	28.22	24.19	25.77	27.43	28.51	23.76	28.01	28.44	29.85	33.69
9	37.45	21.81	35.14	31.62	44.47	35.78	18.38	24.26	100	36.03	32.62	23.92	34.79	32.68	30.68	27.23	33.43	32.56	17.03	29.82
10	34.86	31.31	28.53	28.97	37.33	35.66	14.06	18.07	36.03	100	28.48	29.38	32.29	31.30	29.55	30.80	26.09	28.72	13.18	22.52
11	31.39	18.99	34.61	34.76	31.24	30.78	23.18	28.22	32.62	28.48	100.00	39.05	45.18	41.35	37.06	35.22	26.66	29.23	18.70	29.51
12	27.69	24.37	38.04	50.80	22.32	29.38	18.97	24.19	23.92	29.38	39.05	100.00	49.58	71.65	56.36	48.44	28.91	27.18	14.14	24.61
13	29.08	23.13	38.71	48.33	29.17	34.17	19.77	25.77	34.79	32.29	45.18	49.58	100	50.79	41.46	46.46	32.58	31.41	16.32	27.53
14	29.13	21.46	41.35	53.74	28.94	33.86	20.51	27.43	32.68	31.30	41.35	71.65	50.79	100	57.09	44.88	30.61	31.41	16.17	27.11
15	30.08	24.55	41.61	54.32	27.05	30.00	19.82	28.51	30.68	29.55	37.06	56.36	41.46	57.09	100	48.88	32.44	29.10	15.56	28.88
16	28.88	22.99	36.20	47.10	20.76	29.02	19.13	23.76	27.23	30.80	35.22	48.44	46.46	44.88	48.88	100	27.22	28.08	14.90	27.11
17	31.03	17.63	32.36	31.88	33.29	33.57	22.75	28.01	33.43	26.09	26.66	28.91	32.58	30.61	32.44	27.22	100	26.28	22.55	32.64
18	31.92	16.41	28.85	28.08	35.26	28.33	24.03	28.44	32.56	28.72	29.23	27.18	31.41	31.41	29.10	28.08	26.28	100	22.00	31.91
19	16.73	7.45	24.43	14.14	18.30	14.85	27.37	29.85	17.03	13.18	18.70	14.14	16.32	16.17	15.56	14.90	22.55	22.00	100	30.11
20	30.55	13.14	35.77	26.59	34.41	27.01	32.82	33.69	29.82	22.52	29.51	24.61	27.53	27.11	28.88	27.11	32.64	31.91	30.11	100

Tabel di atas berisi tentang nilai perbandingan antara *source code* data1.java yang dibandingkan dengan data2.java hingga data20.java. Data tersebut menggunakan tahapan *preprocessing,* yaitu *case folding* dan menggunakan algoritma *Levenshtein Distance*. Di tabel terdapat 400 hasil perhitungan yang didapatkan.

Tabel 4.6 *Output* Persentase *Ground Truth* Skenario Kedua

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	100	28.74	34.95	34.15	33.97	42.07	21.10	26.44	46.37	24.55	30.48	37.62	41.14	37.82	32.27	25.89	27.09	33.23	18.99	36.00
2	28.74	100	19.92	34.07	18.36	23.69	9.69	12.56	29.24	39.53	25.71	29.07	24.29	27.63	29.98	27.40	22.77	27.30	9.35	16.23
3	34.95	19.92	100.00	47.49	46.18	33.04	27.64	27.87	31.76	23.02	34.04	37.79	44.30	38.58	53.47	34.02	40.52	37.74	24.91	35.90
4	34.15	34.07	47.49	100.00	27.99	42.63	21.74	22.20	44.07	34.01	47.04	57.64	58.40	62.51	67.33	46.59	41.57	30.20	20.93	35.66
5	33.97	18.36	46.18	27.99	100	31.67	23.45	14.37	45.61	22.28	36.06	23.83	42.45	23.99	31.81	32.65	43.04	30.31	25.76	29.72
6	42.07	23.69	33.04	42.63	31.67	100.00	23.23	23.42	39.25	25.98	35.19	44.55	30.68	48.72	32.65	29.94	38.69	31.19	16.19	36.55
7	21.10	9.69	27.64	21.74	23.45	23.23	100	50.83	24.60	12.60	19.76	21.16	20.53	20.13	27.71	23.57	27.69	20.55	16.73	28.91
8	26.44	12.56	27.87	22.20	14.37	23.42	50.83	100.00	33.17	16.30	21.94	27.13	22.36	24.99	40.90	28.09	27.84	20.29	14.69	31.60
9	46.37	29.24	31.76	44.07	45.61	39.25	24.60	33.17	100	33.65	36.19	34.24	39.86	38.43	36.56	40.19	35.63	37.88	24.61	35.99
10	24.55	39.53	23.02	34.01	22.28	25.98	12.60	16.30	33.65	100.00	24.64	30.21	28.43	26.52	28.09	30.63	22.93	33.54	9.82	21.43
11	30.48	25.71	34.04	47.04	36.06	35.19	19.76	21.94	36.19	24.64	100.00	47.99	48.19	44.96	42.82	41.42	32.45	38.66	24.07	34.99
12	37.62	29.07	37.79	57.64	23.83	44.55	21.16	27.13	34.24	30.21	47.99	100.00	56.15	66.31	69.85	41.04	35.19	28.06	13.93	34.19
13	41.14	24.29	44.30	58.40	42.45	30.68	20.53	22.36	39.86	28.43	48.19	56.15	100	62.35	53.91	56.90	25.23	36.19	18.92	32.52
14	37.82	27.63	38.58	62.51	23.99	48.72	20.13	24.99	38.43	26.52	44.96	66.31	62.35	100	68.57	39.75	36.32	30.75	20.48	35.58
15	32.27	29.98	53.47	67.33	31.81	32.65	27.71	40.90	36.56	28.09	42.82	69.85	53.91	68.57	100	55.63	25.59	34.26	21.88	34.02
16	25.89	27.40	34.02	46.59	32.65	29.94	23.57	28.09	40.19	30.63	41.42	41.04	56.90	39.75	55.63	100	31.11	31.27	15.20	29.28
17	27.09	22.77	40.52	41.57	43.04	38.69	27.69	27.84	35.63	22.93	32.45	35.19	25.23	36.32	25.59	31.11	100	27.80	16.48	36.81
18	33.23	27.30	37.74	30.20	30.31	31.19	20.55	20.29	37.88	33.54	38.66	28.06	36.19	30.75	34.26	31.27	27.80	10	19.40	27.95
19	18.99	9.35	24.91	20.93	25.76	16.19	16.73	14.69	24.61	9.82	24.07	13.93	18.92	20.48	21.88	15.20	16.48	19.40	100	31.65
20	36.00	16.23	35.90	35.66	29.72	36.55	28.91	31.60	35.99	21.43	34.99	34.19	32.52	35.58	34.02	29.28	36.81	27.95	31.65	100

Tabel di atas berisi tentang nilai perbandingan antara *source code* data1.java yang dibandingkan dengan data2.java hingga data20.java. Data tersebut menggunakan tahapan *preprocessing*, yaitu *case folding* dan dihitung secara *Ground Truth*. Di tabel terdapat 400 hasil perhitungan yang didapatkan.

Tabel 4.7 *Confusion Matrix* Skenario Kedua

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	TP	FN	TP	TP	TP	TP	TN	TP	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
2	FN	TP	TN	TP	TN	TN	TN	TN	FN	TP	FN	FN	TN	FN	FN	FN	TN	FN	TN	TN
3	TP	TN	TP	TP	TP	TP	TP	TP	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
4	TP	TP	TP	TP	TP	TP	TN	TN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
5	TP	TN	TP	TP	TP	TP	TN	FP	TP	FP	TP	TN	TP	FP	TP	FN	TP	TP	FN	TP
6	TP	TN	TP	TP	TP	TP	TN	TN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
7	TN	TN	TP	TN	TN	TN	TP	TP	TN	TN	TN	TN	TN	TN	FN	TN	FN	TN	FP	TP
8	TP	TN	TP	TN	FP	TN	TP	TP	FN	TN	FP	FN	FP	FP	TP	FN	TP	FP	FP	TP
9	TP	FN	TP	TP	TP	TP	TN	FN	TP	TP	TP	FN	TP	TP	TP	TP	TP	TP	TN	TP
10	FP	TP	FP	TP	FP	TP	TN	TN	TP	TP	FP	TP	TP	TP	TP	TP	FP	TP	TN	TN
11	TP	FN	TP	TP	TP	TP	TN	FP	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
12	TP	FN	TP	TP	TN	TP	TN	FN	FN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	FN
13	TP	TN	TP	TP	TP	TP	TN	FP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
14	TP	FN	TP	TP	FP	TP	TN	FP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
15	TP	FN	TP	TP	TP	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
16	TP	FN	TP	TP	FN	TP	TN	FN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
17	TP	TN	TP	TP	TP	TP	FN	TP	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
18	TP	FN	TP	TP	TP	TP	TN	FP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
19	TN	TN	TN	TN	FN	TN	FP	FP	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	TP	TP
20	TP	TN	TP	TP	TP	TP	TP	TP	TP	TN	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP

TP	FP	FN	TN
120	13	17	40

Tabel di atas merupakan perhitungan *confusion matrix* yang didapatkan dari perhitungan tabel 4.5 dan tabel 4.6.

Tabel 4.8 *Output* Persentase *Predicted* Skenario Ketiga

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	100	23.11	34.61	29.68	38.45	37.85	20.88	25.20	37.45	34.86	31.39	27.69	29.08	29.13	30.08	28.88	31.03	31.92	16.73	30.55
2	23.11	100	17.31	26.46	20.74	23.26	8.31	10.51	21.81	31.31	18.99	24.37	23.13	21.46	24.55	22.99	17.63	16.41	7.45	13.14
3	34.61	17.31	100	38.44	37.91	32.76	29.36	37.22	35.14	28.53	34.61	38.04	38.71	41.35	41.61	36.20	32.36	28.85	24.43	35.77
4	29.68	26.46	38.44	100	34.56	29.72	17.05	22.53	31.62	28.97	34.76	50.80	48.33	53.74	54.32	47.10	31.88	28.08	14.14	26.59
5	38.45	20.74	37.91	34.56	100	48.62	19.87	25.34	44.47	37.33	31.24	22.32	29.17	28.94	27.05	20.76	33.29	35.26	18.30	34.41
6	37.85	23.26	32.76	29.72	48.62	100	16.78	21.53	35.78	35.66	30.78	29.38	34.17	33.86	30.00	29.02	33.57	28.33	14.85	27.01
7	20.88	8.31	29.36	17.05	19.87	16.78	100	48.53	18.38	14.06	23.18	18.97	19.77	20.51	19.82	19.13	22.75	24.03	27.37	32.82
8	25.20	10.51	37.22	22.53	25.34	21.53	48.53	100.00	24.26	18.07	28.22	24.19	25.77	27.43	28.51	23.76	28.01	28.44	29.85	33.69
9	37.45	21.81	35.14	31.62	44.47	35.78	18.38	24.26	100	36.03	32.62	23.92	34.79	32.68	30.68	27.23	33.43	32.56	17.03	29.82
10	34.86	31.31	28.53	28.97	37.33	35.66	14.06	18.07	36.03	100	28.48	29.38	32.29	31.30	29.55	30.80	26.09	28.72	13.18	22.52
11	31.39	18.99	34.61	34.76	31.24	30.78	23.18	28.22	32.62	28.48	100.00	39.05	45.18	41.35	37.06	35.22	26.66	29.23	18.70	29.51
12	27.69	24.37	38.04	50.80	22.32	29.38	18.97	24.19	23.92	29.38	39.05	100.00	49.58	71.65	56.36	48.44	28.91	27.18	14.14	24.61
13	29.08	23.13	38.71	48.33	29.17	34.17	19.77	25.77	34.79	32.29	45.18	49.58	100	50.79	41.46	46.46	32.58	31.41	16.32	27.53
14	29.13	21.46	41.35	53.74	28.94	33.86	20.51	27.43	32.68	31.30	41.35	71.65	50.79	100	57.09	44.88	30.61	31.41	16.17	27.11
15	30.08	24.55	41.61	54.32	27.05	30.00	19.82	28.51	30.68	29.55	37.06	56.36	41.46	57.09	100	48.88	32.44	29.10	15.56	28.88
16	28.88	22.99	36.20	47.10	20.76	29.02	19.13	23.76	27.23	30.80	35.22	48.44	46.46	44.88	48.88	100	27.22	28.08	14.90	27.11
17	31.03	17.63	32.36	31.88	33.29	33.57	22.75	28.01	33.43	26.09	26.66	28.91	32.58	30.61	32.44	27.22	100	26.28	22.55	32.64
18	31.92	16.41	28.85	28.08	35.26	28.33	24.03	28.44	32.56	28.72	29.23	27.18	31.41	31.41	29.10	28.08	26.28	100	22.00	31.91
19	16.73	7.45	24.43	14.14	18.30	14.85	27.37	29.85	17.03	13.18	18.70	14.14	16.32	16.17	15.56	14.90	22.55	22.00	100	30.11
20	30.55	13.14	35.77	26.59	34.41	27.01	32.82	33.69	29.82	22.52	29.51	24.61	27.53	27.11	28.88	27.11	32.64	31.91	30.11	100

Tabel di atas berisi tentang nilai perbandingan antara *source code* data1.java yang dibandingkan dengan data2.java hingga data20.java. Data tersebut menggunakan tahapan *preprocessing*, yaitu *filtering* dan menggunakan algoritma *Levenshtein Distance*. Di tabel terdapat 400 hasil perhitungan yang didapatkan.

Tabel 4.9 *Output* Persentase *Ground Truth* Skenario Ketiga

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	100	28.74	34.95	34.15	33.97	42.07	21.10	26.44	46.37	24.55	30.48	37.62	41.14	37.82	32.27	25.89	27.09	33.23	18.99	36.00
2	28.74	100	19.92	34.07	18.36	23.69	9.69	12.56	29.24	39.53	25.71	29.07	24.29	27.63	29.98	27.40	22.77	27.30	9.35	16.23
3	34.95	19.92	100.00	47.49	46.18	33.04	27.64	27.87	31.76	23.02	34.04	37.79	44.30	38.58	53.47	34.02	40.52	37.74	24.91	35.90
4	34.15	34.07	47.49	100.00	27.99	42.63	21.74	22.20	44.07	34.01	47.04	57.64	58.40	62.51	67.33	46.59	41.57	30.20	20.93	35.66
5	33.97	18.36	46.18	27.99	100	31.67	23.45	14.37	45.61	22.28	36.06	23.83	42.45	23.99	31.81	32.65	43.04	30.31	25.76	29.72
6	42.07	23.69	33.04	42.63	31.67	100.00	23.23	23.42	39.25	25.98	35.19	44.55	30.68	48.72	32.65	29.94	38.69	31.19	16.19	36.55
7	21.10	9.69	27.64	21.74	23.45	23.23	100	50.83	24.60	12.60	19.76	21.16	20.53	20.13	27.71	23.57	27.69	20.55	16.73	28.91
8	26.44	12.56	27.87	22.20	14.37	23.42	50.83	100.00	33.17	16.30	21.94	27.13	22.36	24.99	40.90	28.09	27.84	20.29	14.69	31.60
9	46.37	29.24	31.76	44.07	45.61	39.25	24.60	33.17	100	33.65	36.19	34.24	39.86	38.43	36.56	40.19	35.63	37.88	24.61	35.99
10	24.55	39.53	23.02	34.01	22.28	25.98	12.60	16.30	33.65	100.00	24.64	30.21	28.43	26.52	28.09	30.63	22.93	33.54	9.82	21.43
11	30.48	25.71	34.04	47.04	36.06	35.19	19.76	21.94	36.19	24.64	100.00	47.99	48.19	44.96	42.82	41.42	32.45	38.66	24.07	34.99
12	37.62	29.07	37.79	57.64	23.83	44.55	21.16	27.13	34.24	30.21	47.99	100.00	56.15	66.31	69.85	41.04	35.19	28.06	13.93	34.19
13	41.14	24.29	44.30	58.40	42.45	30.68	20.53	22.36	39.86	28.43	48.19	56.15	100	62.35	53.91	56.90	25.23	36.19	18.92	32.52
14	37.82	27.63	38.58	62.51	23.99	48.72	20.13	24.99	38.43	26.52	44.96	66.31	62.35	100	68.57	39.75	36.32	30.75	20.48	35.58
15	32.27	29.98	53.47	67.33	31.81	32.65	27.71	40.90	36.56	28.09	42.82	69.85	53.91	68.57	100	55.63	25.59	34.26	21.88	34.02
16	25.89	27.40	34.02	46.59	32.65	29.94	23.57	28.09	40.19	30.63	41.42	41.04	56.90	39.75	55.63	100	31.11	31.27	15.20	29.28
17	27.09	22.77	40.52	41.57	43.04	38.69	27.69	27.84	35.63	22.93	32.45	35.19	25.23	36.32	25.59	31.11	100	27.80	16.48	36.81
18	33.23	27.30	37.74	30.20	30.31	31.19	20.55	20.29	37.88	33.54	38.66	28.06	36.19	30.75	34.26	31.27	27.80	10	19.40	27.95
19	18.99	9.35	24.91	20.93	25.76	16.19	16.73	14.69	24.61	9.82	24.07	13.93	18.92	20.48	21.88	15.20	16.48	19.40	100	31.65
20	36.00	16.23	35.90	35.66	29.72	36.55	28.91	31.60	35.99	21.43	34.99	34.19	32.52	35.58	34.02	29.28	36.81	27.95	31.65	100

Tabel di atas berisi tentang nilai perbandingan antara *source code* data1.java yang dibandingkan dengan data2.java hingga data20.java. Data tersebut menggunakan tahapan *preprocessing*, yaitu *filtering* dan dihitung secara *Ground Truth*. Di tabel terdapat 400 hasil perhitungan yang didapatkan.

Tabel 4.10 *Confusion Matrix* Skenario Ketiga

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	TP	TN	TP	TP	TP	TP	TN	TN	TP	FP	TP	FN	TP	TP	TP	FN	TP	TP	TN	TP
2	TN	TP	TN	FN	TN	TN	TN	TN	TN	FN	TN	FN	TN	TN	TN	TN	TN	TN	TN	TN
3	TP	TN	TP	TP	TP	TP	TP	TP	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
4	TP	FN	TP	TP	TP	TP	TN	FN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	FN
5	TP	TN	TP	TP	TP	TP	TN	FP	TP	FP	TP	TN	TP	FP	TP	FN	TP	TP	TN	TP
6	TP	TN	TP	TP	TP	TP	TN	TN	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
7	TN	TN	TP	TN	TN	TN	TP	TP	TN	TN	TN	TN	TN	TN	TN	TN	FN	TN	FP	FP
8	TN	TN	TP	FN	FP	TN	TP	TP	FN	TN	FP	FN	TN	FP	TP	FN	TP	FP	FP	FP
9	TP	TN	TP	TP	TP	TP	TN	FN	TP	TP	TP	FN	TP	TP	TP	FN	TP	TP	TN	TP
10	FP	FN	FP	TP	FP	FP	TN	TN	TP	TP	FP	TP	FP	FP	FP	TP	FP	TP	TN	TN
11	TP	TN	TP	TP	TP	TP	TN	FP	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
12	FN	FN	TP	TP	TN	TP	TN	FN	FN	TP	TP	TP	TP	TP	TP	TP	TP	FN	TN	FN
13	TP	TN	TP	TP	TP	TP	TN	TN	TP	FP	TP	TP	TP	TP	TP	TP	FP	TP	TN	TP
14	TP	TN	TP	TP	FP	TP	TN	FP	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
15	TP	TN	TP	TP	TP	TP	TN	TP	TP	FP	TP	TP	TP	TP	TP	TP	FP	TP	TN	TP
16	FN	TN	TP	TP	FN	TP	TN	FN	FN	TP	TP	TP	TP	TP	TP	TP	FN	TP	TN	FN
17	TP	TN	TP	TP	TP	TP	FN	TP	TP	FP	TP	TP	FP	TP	FP	FN	TP	TP	TN	TP
18	TP	TN	TP	TP	TP	TP	TN	FP	TP	TP	TP	FN	TP	TP	TP	TP	TP	TP	TN	TP
19	TN	TN	TN	TN	TN	TN	FP	FP	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	TP	TP
20	TP	TN	TP	FN	TP	TP	FP	FP	TP	TN	TP	FN	TP	TP	TP	FN	TP	TP	TP	TP

TP	FP	FN	TN
102	20	18	50

Tabel di atas merupakan perhitungan *confusion matrix* yang didapatkan dari perhitungan tabel 4.8 dan tabel 4.9.

Tabel 4.11 *Output* Persentase *Predicted* Skenario Keempat

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	100	19.43	32.27	28.76	39.12	37.56	18.95	23.46	32.64	33.94	29.90	20.21	27.08	31.79	22.28	22.68	29.89	31.22	14.30	28.84
2	19.43	100.	14.92	23.59	19.84	21.64	6.84	8.85	17.88	24.00	15.37	20.57	18.75	17.44	21.58	19.59	14.47	12.52	6.04	11.02
3	32.27	14.92	100.00	35.92	37.29	32.42	27.72	36.20	33.49	27.70	35.77	36.07	36.38	39.27	39.73	32.72	30.59	29.60	22.57	35.29
4	28.76	23.59	35.92	100.00	36.19	33.63	15.67	21.02	32.12	31.23	32.60	47.92	46.30	51.43	50.00	40.98	28.93	25.92	12.13	24.15
5	39.12	19.84	37.29	36.19	100	46.92	18.77	25.00	42.90	37.80	30.57	23.44	30.32	30.68	29.21	21.65	34.34	34.17	16.69	33.88
6	37.56	21.64	32.42	33.63	46.92	100.00	15.91	21.02	34.92	36.26	29.05	27.86	32.18	34.00	30.26	25.00	32.91	27.39	13.70	26.38
7	18.95	6.84	27.72	15.67	18.77	15.91	100	47.49	17.13	11.99	23.04	17.66	18.89	19.24	18.36	17.25	21.75	23.10	27.90	33.27
8	23.46	8.85	36.20	21.02	25.00	21.02	47.49	100.00	22.97	15.75	28.81	22.81	24.68	25.89	27.11	21.75	27.52	28.25	29.47	35.23
9	32.64	17.88	33.49	32.12	42.90	34.92	17.13	22.97	100	37.15	30.07	21.61	31.25	29.58	26.84	22.16	31.96	31.81	15.39	28.14
10	33.94	24.00	27.70	31.23	37.80	36.26	11.99	15.75	37.15	100.00	28.04	27.86	31.71	29.80	29.47	29.12	25.28	26.80	11.42	20.98
11	29.90	15.37	35.77	32.60	30.57	29.05	23.04	28.81	30.07	28.04	100.00	36.49	41.55	38.34	34.12	32.09	26.39	29.46	17.78	27.78
12	20.21	20.57	36.07	47.92	23.44	27.86	17.66	22.81	21.61	27.86	36.49	100.00	45.83	68.21	53.91	45.62	26.71	25.18	12.72	22.51
13	27.08	18.75	36.38	46.30	30.32	32.18	18.89	24.68	31.25	31.71	41.55	45.83	100	47.24	38.66	44.44	29.89	30.78	15.12	25.56
14	31.79	17.44	39.27	51.43	30.68	34.00	19.24	25.89	29.58	29.80	38.34	68.21	47.24	100	55.85	42.83	28.14	29.46	14.63	24.97
15	22.28	21.58	39.73	50.00	29.21	30.26	18.36	27.11	26.84	29.47	34.12	53.91	38.66	55.85	100	42.01	30.21	27.98	13.59	25.91
16	22.68	19.59	32.72	40.98	21.65	25.00	17.25	21.75	22.16	29.12	32.09	45.62	44.44	42.83	42.01	100	24.32	26.07	13.10	23.92
17	29.89	14.47	30.59	28.93	34.34	32.91	21.75	27.52	31.96	25.28	26.39	26.71	29.89	28.14	30.21	24.32	100	26.80	21.42	31.77
18	31.22	12.52	29.60	25.92	34.17	27.39	23.10	28.25	31.81	26.80	29.46	25.18	30.78	29.46	27.98	26.07	26.80	100	21.42	31.77
19	14.30	6.04	22.57	12.13	16.69	13.70	27.90	29.47	15.39	11.42	17.78	12.72	15.12	14.63	13.59	13.10	21.42	21.42	100	30.23
20	28.84	11.02	35.29	24.15	33.88	26.38	33.27	35.23	28.14	20.98	27.78	22.51	25.56	24.97	25.91	23.92	31.77	31.77	30.23	100

Tabel di atas berisi tentang nilai perbandingan antara *source code* data1.java yang dibandingkan dengan data2.java hingga data20.java. Data tersebut menggunakan tahapan *preprocessing*, yaitu *case folding* dan *filtering* menggunakan algoritma *Levenshtein Distance*. Di tabel terdapat 400 hasil perhitungan yang didapatkan.

Tabel 4.12 *Output* Persentase *Ground Truth* Skenario Keempat

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	100	23.37	31.83	31.15	33.47	38.46	18.99	25.09	41.13	20.44	26.79	33.77	34.47	34.09	28.98	24.81	24.43	30.61	18.52	33.41
2	23.37	100	15.13	25.63	13.75	19.25	9.75	11.99	21.46	24.35	17.31	27.69	24.65	23.77	22.87	19.85	17.25	19.88	6.99	11.73
3	31.83	15.13	100.00	44.47	45.63	31.23	26.02	26.15	34.09	22.27	30.74	34.97	42.42	36.22	51.30	30.62	40.12	37.57	22.60	34.17
4	31.15	25.63	44.47	100.00	24.33	40.44	17.90	19.18	40.06	34.85	47.48	53.14	55.66	60.21	60.50	41.51	29.03	28.98	17.01	32.24
5	33.47	13.75	45.63	24.33	100	27.13	22.37	12.59	44.60	19.26	34.61	19.55	39.50	21.31	27.09	27.33	43.11	30.42	24.50	27.90
6	38.46	19.25	31.23	40.44	27.13	100.00	20.76	22.36	42.00	21.62	32.33	41.05	27.39	44.78	31.58	27.40	37.90	29.97	16.69	34.81
7	18.99	9.75	26.02	17.90	22.37	20.76	100	48.95	19.15	11.12	19.90	18.91	18.30	17.94	23.83	20.88	27.02	18.42	12.57	24.81
8	25.09	11.99	26.15	19.18	12.59	22.36	48.95	100.00	25.28	14.57	20.07	24.88	19.95	22.79	39.33	24.69	26.76	19.57	13.48	24.08
9	41.13	21.46	34.09	40.06	44.60	42.00	19.15	25.28	100	27.30	31.37	30.19	35.44	35.02	33.06	35.39	32.83	28.74	18.84	28.24
10	20.44	24.35	22.27	34.85	19.26	21.62	11.12	14.57	27.30	100.00	32.30	24.61	36.36	25.89	22.86	29.15	19.57	27.99	16.47	18.13
11	26.79	17.31	30.74	47.48	34.61	32.33	19.90	20.07	31.37	32.30	100.00	44.47	47.85	43.83	41.77	41.43	28.99	37.45	22.46	33.22
12	33.77	27.69	34.97	53.14	19.55	41.05	18.91	24.88	30.19	24.61	44.47	100.00	52.21	63.56	68.59	36.79	31.19	26.34	11.34	31.04
13	34.47	24.65	42.42	55.66	39.50	27.39	18.30	19.95	35.44	36.36	47.85	52.21	100	59.44	50.99	54.39	20.92	32.94	18.23	29.73
14	34.09	23.77	36.22	60.21	21.31	44.78	17.94	22.79	35.02	25.89	43.83	63.56	59.44	100	67.23	35.91	36.41	31.80	16.49	32.31
15	28.98	22.87	51.30	60.50	27.09	31.58	23.83	39.33	33.06	22.86	41.77	68.59	50.99	67.23	100	51.82	21.21	31.73	16.76	28.55
16	24.81	19.85	30.62	41.51	27.33	27.40	20.88	24.69	35.39	29.15	41.43	36.79	54.39	35.91	51.82	100	27.14	29.62	13.83	25.79
17	24.43	17.25	40.12	29.03	43.11	37.90	27.02	26.76	32.83	19.57	28.99	31.19	20.92	36.41	21.21	27.14	100	25.38	14.67	35.49
18	30.61	19.88	37.57	28.98	30.42	29.97	18.42	19.57	28.74	27.99	37.45	26.34	32.94	31.80	31.73	29.62	25.38	100.	18.98	32.25
19	18.52	6.99	22.60	17.01	24.50	16.69	12.57	13.48	18.84	16.47	22.46	11.34	18.23	16.49	16.76	13.83	14.67	18.98	100	28.90
20	33.41	11.73	34.17	32.24	27.90	34.81	24.81	24.08	28.24	18.13	33.22	31.04	29.73	32.31	28.55	25.79	35.49	32.25	28.90	100

Tabel di atas berisi tentang nilai perbandingan antara *source code* data1.java yang dibandingkan dengan data2.java hingga data20.java. Data tersebut menggunakan tahapan *preprocessing,* yaitu *case folding* dan *filtering* menggunakan *Ground Truth*. Di tabel terdapat 400 hasil perhitungan yang didapatkan.

Tabel 4.13 *Confusion Matrix* Skenario Keempat

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	TP	TN	TP	TP	TP	TP	TN	FN	TP	FP	TP	FN	TP	TP	FN	TN	FP	TP	TN	TP
2	TN	TP	TN	FN	TN	TN	TN	TN	TN	TN	TN	FN	TN	TN	TN	TN	TN	TN	TN	TN
3	TP	TN	TP	TP	TP	TP	TP	TP	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
4	TP	FN	TP	TP	FP	TP	TN	TN	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	FN
5	TP	TN	TP	FP	TP	TP	TN	FP	TP	FP	TP	TN	TP	FP	TP	FN	TP	TP	TN	TP
6	TP	TN	TP	TP	TP	TP	TN	TN	TP	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
7	TN	TN	TP	TN	TN	TN	TP	TP	TN	TN	TN	TN	TN	TN	TN	TN	FN	TN	FP	FP
8	FN	TN	TP	TN	FP	TN	TP	TP	FN	TN	FP	TN	TN	FP	TP	TN	TP	FP	FP	FP
9	TP	TN	TP	TP	TP	TP	TN	FN	TP	TP	TP	FN	TP	TP	TP	FN	TP	TP	TN	TP
10	FP	TN	FP	TP	FP	FP	TN	TN	TP	TP	TP	FP	TP	TP	FP	TP	FP	TP	TN	TN
11	TP	TN	TP	TP	TP	TP	TN	FP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
12	FN	FN	TP	TP	TN	TP	TN	TN	FN	FP	TP	TP	TP	TP	TP	TP	TP	TP	TN	FN
13	TP	TN	TP	TP	TP	TP	TN	TN	TP	TP	TP	TP	TP	TP	TP	TP	FP	TP	TN	TP
14	TP	TN	TP	TP	FP	TP	TN	FP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	FN
15	FN	TN	TP	TP	TP	TP	TN	TP	TP	FP	TP	TP	TP	TP	TP	TP	FP	TP	TN	TP
16	TN	TN	TP	TP	FN	TP	TN	TN	FN	TP	TP	TP	TP	TP	TP	TP	FN	TP	TN	FN
17	FP	TN	TP	TP	TP	TP	FN	TP	TP	FP	TP	TP	FP	TP	FP	FN	TP	TP	TN	TP
18	TP	TN	TP	TP	TP	TP	TN	FP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	TN	TP
19	TN	TN	TN	TN	TN	TN	FP	FP	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	TP	TP
	TP	TN	TP	FN	TP	TP	FP	FP	TP	TN	TP	FN	TP	FN	TP	FN	TP	TP	TP	TP

TP	FP	FN	TN
101	20	15	54

Tabel di atas merupakan perhitungan *confusion matrix* yang didapatkan dari perhitungan tabel 4.12 dan tabel 4.13.

Tabel 4.14 Perhitungan Skenario Pertama

Nama Data	Predicted		Ground Truth		TP	TN	FP	FN
	Persentase	Keterangan	Persentase	Keterangan				
data2.java	23.0616%	TIDAK PLAGIAT	28.1159%	PLAGIAT	0	0	0	1
data3.java	34.2140%	PLAGIAT	34.7619%	PLAGIAT	1	0	0	0
data4.java	29.2247%	PLAGIAT	33.1787%	PLAGIAT	1	0	0	0
data5.java	38.1710%	PLAGIAT	33.9381%	PLAGIAT	1	0	0	0
data6.java	36.7793%	PLAGIAT	41.5730%	PLAGIAT	1	0	0	0
data7.java	20.4582%	TIDAK PLAGIAT	20.7563%	TIDAK PLAGIAT	0	1	0	0
data8.java	24.6940%	TIDAK PLAGIAT	26.2156%	PLAGIAT	0	0	0	1
data9.java	36.9781%	PLAGIAT	45.8836%	PLAGIAT	1	0	0	0
data10.java	34.3936%	PLAGIAT	24.2788%	TIDAK PLAGIAT	0	0	1	0
data11.java	30.6279%	PLAGIAT	30.2768%	PLAGIAT	1	0	0	0
data12.java	27.0378%	PLAGIAT	36.9427%	PLAGIAT	1	0	0	0

data13.java	28.6282%	PLAGIAT	40.6918%	PLAGIAT	1	0	0	0
data14.java	28.3465%	PLAGIAT	36.9931%	PLAGIAT	1	0	0	0
data15.java	29.0258%	PLAGIAT	31.6013%	PLAGIAT	1	0	0	0
data16.java	28.2306%	PLAGIAT	25.4469%	PLAGIAT	1	0	0	0
data17.java	30.3244%	PLAGIAT	26.0726%	PLAGIAT	1	0	0	0
data18.java	31.1538%	PLAGIAT	32.8917%	PLAGIAT	1	0	0	0
data19.java	16.1176%	TIDAK PLAGIAT	18.9822%	TIDAK PLAGIAT	0	1	0	0
data20.java	29.7185%	PLAGIAT	34.4733%	PLAGIAT	1	0	0	0
				Jumlah	14	2	1	2

Tabel di atas merupakan perhitungan mendetail terkait data1.java dibandingkan data2.java hingga data20.java menggunakan skenario pertama, yaitu menggunakan algoritma *Levenshtein Distance* autentik.

Tabel 4.15 Perhitungan Skenario Kedua

Nama Data	Predicted		Ground Truth		TP	TN	FP	FN
	Persentase	Keterangan	Persentase	Keterangan				
data2.java	23.0616%	TIDAK PLAGIAT	28.6957%	PLAGIAT	0	0	0	1
data3.java	34.4782%	PLAGIAT	34.9206%	PLAGIAT	1	0	0	0
data4.java	29.6223%	PLAGIAT	34.1067%	PLAGIAT	1	0	0	0
data5.java	38.3698%	PLAGIAT	33.9381%	PLAGIAT	1	0	0	0
data6.java	37.7734%	PLAGIAT	42.0225%	PLAGIAT	1	0	0	0
data7.java	20.8311%	TIDAK PLAGIAT	21.0924%	TIDAK PLAGIAT	0	1	0	0
data8.java	25.1260%	PLAGIAT	26.4271%	PLAGIAT	1	0	0	0
data9.java	37.3757%	PLAGIAT	46.3227%	PLAGIAT	1	0	0	0
data10.java	34.7913%	PLAGIAT	24.5192%	TIDAK PLAGIAT	0	0	1	0
data11.java	31.3936%	PLAGIAT	30.4498%	PLAGIAT	1	0	0	0
data12.java	27.8330%	PLAGIAT	37.5796%	PLAGIAT	1	0	0	0

data13.java	29.2247%	PLAGIAT	41.0987%	PLAGIAT	1	0	0	0
data14.java	29.1339%	PLAGIAT	37.7844%	PLAGIAT	1	0	0	0
data15.java	30.0199%	PLAGIAT	32.2375%	PLAGIAT	1	0	0	0
data16.java	28.8270%	PLAGIAT	25.8675%	PLAGIAT	1	0	0	0
data17.java	30.8886%	PLAGIAT	27.0627%	PLAGIAT	1	0	0	0
data18.java	31.7949%	PLAGIAT	33.2034%	PLAGIAT	1	0	0	0
data19.java	16.6751%	TIDAK PLAGIAT	18.9822%	TIDAK PLAGIAT	0	1	0	0
data20.java	30.5527%	PLAGIAT	35.9781%	PLAGIAT	1	0	0	0
Jumlah					15	2	1	1

Tabel di atas merupakan perhitungan mendetail terkait data1.java dibandingkan data2.java hingga data20.java menggunakan skenario kedua, yaitu menggunakan algoritma *Levenshtein Distance* autentik ditambah tahap *preprocessing case folding*.

Tabel 4.16 Perhitungan Skenario Ketiga

Nama Data	Predicted		Ground Truth		TP	TN	FP	FN
	Persentase	Keterangan	Persentase	Keterangan				
data2.java	20.2381%	TIDAK PLAGIAT	24.6914%	TIDAK PLAGIAT	0	1	0	0
data3.java	33.2851%	PLAGIAT	32.0432%	PLAGIAT	1	0	0	0
data4.java	28.3333%	PLAGIAT	30.6849%	PLAGIAT	1	0	0	0
data5.java	38.8095%	PLAGIAT	32.7910%	PLAGIAT	1	0	0	0
data6.java	35.9524%	PLAGIAT	38.2813%	PLAGIAT	1	0	0	0
data7.java	18.9705%	TIDAK PLAGIAT	18.7994%	TIDAK PLAGIAT	0	1	0	0
data8.java	23.6568%	TIDAK PLAGIAT	24.9550%	TIDAK PLAGIAT	0	1	0	0
data9.java	32.8571%	PLAGIAT	41.0714%	PLAGIAT	1	0	0	0
data10.java	33.3333%	PLAGIAT	21.0059%	TIDAK PLAGIAT	0	0	1	0
data11.java	29.1667%	PLAGIAT	27.4510%	PLAGIAT	1	0	0	0
data12.java	22.6190%	TIDAK PLAGIAT	33.6609%	PLAGIAT	0	0	0	1

data13.java	24.7727%	TIDAK PLAGIAT	34.1860%	PLAGIAT	0	0	0	1
data14.java	30.0216%	PLAGIAT	34.2016%	PLAGIAT	1	0	0	0
data15.java	25.2381%	PLAGIAT	28.5714%	PLAGIAT	1	0	0	0
data16.java	22.8571%	TIDAK PLAGIAT	25.1220%	PLAGIAT	0	0	0	1
data17.java	29.4210%	PLAGIAT	25.1180%	PLAGIAT	1	0	0	0
data18.java	29.9270%	PLAGIAT	31.3122%	PLAGIAT	1	0	0	0
data19.java	14.4081%	TIDAK PLAGIAT	18.1898%	TIDAK PLAGIAT	0	1	0	0
data20.java	28.7370%	PLAGIAT	33.2034%	PLAGIAT	1	0	0	0
				Jumlah	11	4	1	3

Tabel di atas merupakan perhitungan mendetail terkait data1.java dibandingkan data2.java hingga data20.java menggunakan skenario ketiga, yaitu menggunakan algoritma *Levenshtein Distance* autentik ditambah tahap *preprocessing filtering*.

Tabel 4.17 Perhitungan Skenario Keempat

Nama Data	Predicted		Ground Truth		TP	TN	FP	FN
	Persentase	Keterangan	Persentase	Keterangan				
data2.java	19.3798%	TIDAK PLAGIAT	23.3270%	TIDAK PLAGIAT	0	1	0	0
data3.java	32.4201%	PLAGIAT	31.9923%	PLAGIAT	1	0	0	0
data4.java	28.6822%	PLAGIAT	31.3953%	PLAGIAT	1	0	0	0
data5.java	39.0181%	PLAGIAT	33.4211%	PLAGIAT	1	0	0	0
data6.java	37.4677%	PLAGIAT	38.4088%	PLAGIAT	1	0	0	0
data7.java	19.0058%	TIDAK PLAGIAT	19.0749%	TIDAK PLAGIAT	0	1	0	0
data8.java	23.5390%	TIDAK PLAGIAT	25.2007%	PLAGIAT	0	0	0	1
data9.java	32.5581%	PLAGIAT	41.0738%	PLAGIAT	1	0	0	0
data10.java	33.8501%	PLAGIAT	20.4082%	TIDAK PLAGIAT	0	0	1	0
data11.java	29.8986%	PLAGIAT	26.7620%	PLAGIAT	1	0	0	0
data12.java	20.1550%	TIDAK PLAGIAT	33.7224%	PLAGIAT	0	0	0	1

data13.java	26.8519%	PLAGIAT	34.4322%	PLAGIAT	1	0	0	0
data14.java	31.7881%	PLAGIAT	34.0476%	PLAGIAT	1	0	0	0
data15.java	22.7390%	TIDAK PLAGIAT	29.2047%	PLAGIAT	0	0	0	1
data16.java	22.4227%	TIDAK PLAGIAT	25.0323%	PLAGIAT	0	0	0	1
data17.java	30.0477%	PLAGIAT	24.6063%	TIDAK PLAGIAT	0	0	1	0
data18.java	31.0751%	PLAGIAT	30.5816%	PLAGIAT	1	0	0	0
data19.java	14.3556%	TIDAK PLAGIAT	18.5984%	TIDAK PLAGIAT	0	1	0	0
data20.java	28.9566%	PLAGIAT	33.5484%	PLAGIAT	1	0	0	0

10	3	2	4

Tabel di atas merupakan perhitungan mendetail terkait data1.java dibandingkan data2.java hingga data20.java menggunakan skenario keempat, yaitu menggunakan algoritma *Levenshtein Distance* autentik ditambah tahap *preprocessing case folding* dan *filterin*

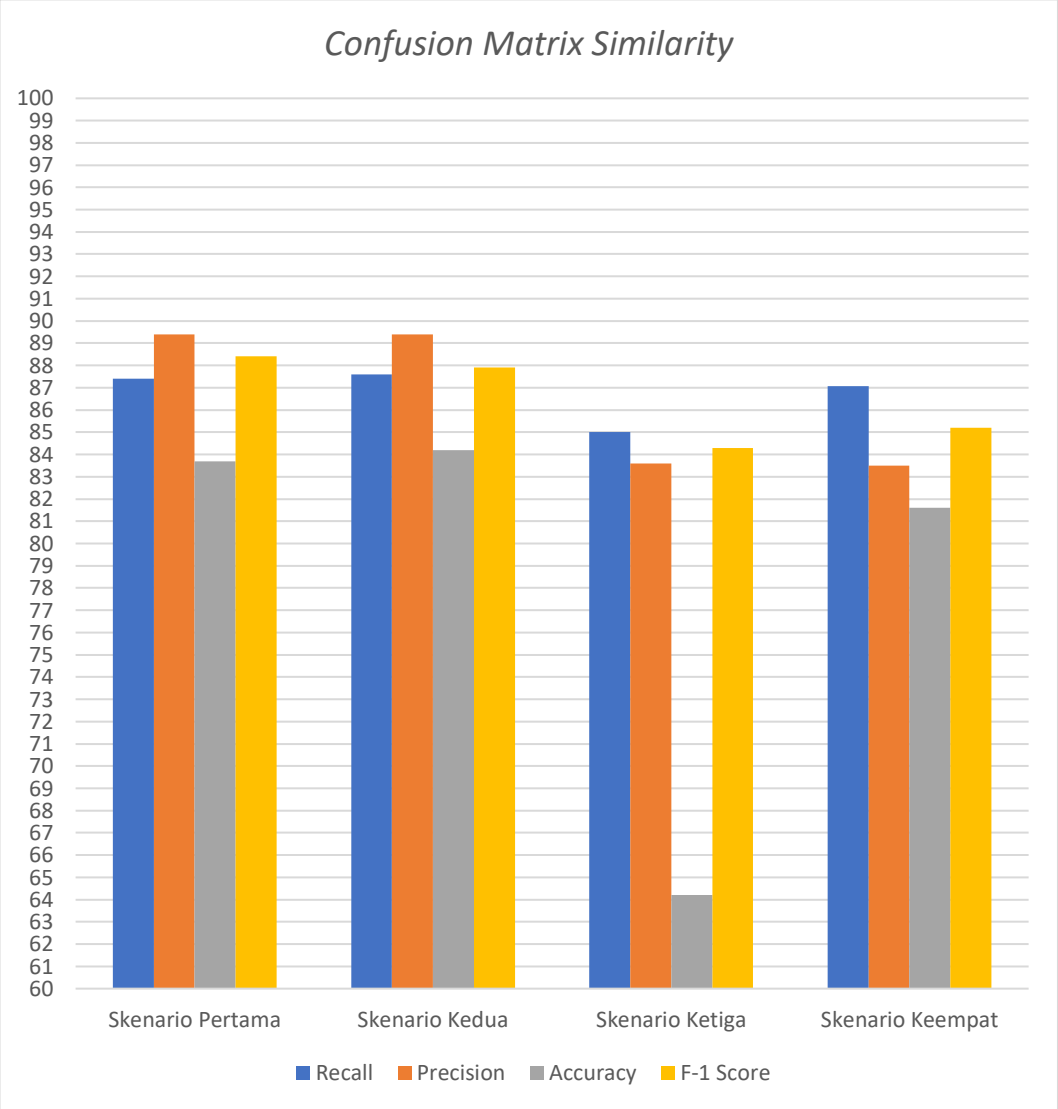
Tabel 4.18 Perhitungan Nilai *Confusion Matrix*

Skenario	**Perhitungan**			
	Accuracy	***Precision***	***Recall***	***F1-Score***
Skenario Pertama	83,7%	89,4%	87,4%	88,4%
Skenario Kedua	84,2%	89,4%	87,6%	87,9%
Skenario Ketiga	64,2%	83,6%	85%	84,3%
Skenario Keempat	81,6%	83,5%	87,06%	85,2%

4.3 Pembahasan

Setelah peneliti melakukan percobaan uji coba perbandingan antara *source code* bahasa pemrograman Java satu dan lainnya dengan menggunakan metode *Levenshtein Distance*, dan dari beberapa skenario yang ada, maka didapatkan hasil dari skenario pertama yang berupa *Levenshtein Distance* autentik, yaitu *recall* sebesar 87,4%, *precision* 89,4%, *accuracy* 83,7%, dan F_1-*score* 88,4%. Adapun skenario kedua yang berupa *Levenshtein Distance* dengan tahaan *preprocessing case folding*, maka didapatkan hasil *recall* 87,6%, *precision* 90,2%, *accuracy* 84,2%, dan F_1-*Score* 87,9%. Sedangkan pada skenario ketiga yang menerapkan *Levenshtein Distance* ditambah tahapan *preprocesing filtering*, didapatkan hasil *recall* 85%, *precison* 83,6%, *accuracy* 64,2%, dan F_1-*score* 84,3%. Sedangkan pada skenario terakhir, yang berupa *Levenshtein Distance* dan menggunakan tahapan *preprocessing* berupa *case folding* dan *filtering*, didapatkan hasil *recall* 87,06%, *precision* 83,5%, *accuracy* 81,6%, dan F_1-*score* sebesar 85,2%. Hasil tersebut tergambarkan seperti dalam bar grafik di bawah ini.

Dari penelitian yang telah dilakukan dan paparan hasil di bab sebelumnya, maka didapatkan hasil dari tiap perhitungan, yaitu *recall* pada skenario kedua sebesar 87,6%, *precision* pada skenario kedua sebesar 90,2%, *accuracy* pada skenario kedua sebesar 84,2%, sedangan *F-1 Score* pada skenario pertama, yaitu sebesar 88,4%.



Gambar 4.5 Bar Grafik *Confusion Matrix Similarity*

Dari hasil di atas, didapatkan hasil yang didominasi di atas 80 persen. Hal itu disebabkan banyak hal, salah satunya seperti *dataset* yang masih terlalu

sedikit *line*-nya, sehingga didapatkan kemiripan yang banyak, dan ketika dimasukkan ke dalam tahapan *confusion matrix*, maka *predicted* dan *Ground Truth* menghasilkan hasil yang sama. Selain itu, menurut penulis, akibat *dataset* yang telah dikumpulkan penulis tidak dalam satu tugas yang sama dari para praktikkan, maka persentase akhir yang dihasilkan juga berpengaruh signifikan.

Sistem deteksi plagiarisme pada *source code* Java yang dibuat berdasarkan *web-based* ditujukan agar *copy-paste* di antara praktikan bisa lebih diminimalisir, terlebih lagi dalam dunia akademis. Meng-*copy paste* dengan atau tanpa sengaja dengan batasan ambang batas tertentu dapat dikategorikan sebagai pengambil atau penjiplakan tugas oleh praktikan. Jika dikaitkan dengan implementasi dan integrasi dalam Islam, hal ini senada dengan firman Allah di surah Asy-Syu'ara' ayat 183, yang berbunyi:

وَلَا تَبْخَسُوا النَّاسَ أَشْيَاءَهُمْ وَلَا تَعْثَوْا فِي الْأَرْضِ مُفْسِدِينَ

Dan janganlah kamu merugikan manusia pada hak-haknya dan janganlah kamu merajalela di muka bumi dengan membuat kerusakan.(Asy-Syuara: 183)

Ayat di atas sebagai pengingat kepada muslim agar tidak bermudah-mudah mengambil hak orang lain, terlebih lagi kepada muslim yang lain, terutama di dunia akademik yang lebih banyak membutuhkan integritas di dalamnya.

Lembaga Fatwa Mesir, Darul Ifta Al-Mishriyyah di laman *website*-nya juga melansir pendapat terkait plagiarisme secara luas.

حقوق التأليف والاختراع أو الابتكار مصونة شرعا، ولأصحابها حق التصرف فيها، ولا يجوز

الاعتداء عليها والله أعلم. وبناء على ذلك: فإن انتحال الحقوق الفكرية والعلامات التجارية المسجلة

لأصحابها بطريقة يفهم بها المنتحل الناس أنها العلامة الأصلية هو أمر محرم شرعا يدخل في باب الكذب

والغش والتدليس، وفيه تضييع لحقوق الناس وأكل لأموالهم بالباطل

Hak karya tulis dan karya-karya kreatif, dilindungi secara syara'. Pemiliknya memunyai hak pendayagunaan karya-karya tersebut. Siapapun tidak boleh berlaku zalim terhadap hak mereka. Berdasarkan pendapat ini, kejahatan plagiasi terhadap hak intelektual dan hak merk dagang yang terregistrasi dengan cara mengakui karya tersebut di hadapan publik, merupakan tindakan yang diharamkan syara'. Kasus ini masuk dalam larangan dusta, pemalsuan, penggelapan. Pada kasus ini, terdapat praktik penelantaran terhadap hak orang lain; dan praktik memakan harta orang lain dengan cara batil.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari penelitian yang telah dilakukan dan paparan hasil di bab sebelumnya, maka didapatkan hasil yang paling signifikan didapatkan pada skenario kedua, yaitu berupa penghitungan dengan algoritma *Levenshtein Distance* ditambah tahapan *preprocessing case folding*, yaitu *accuracy* sebesar 84,2%, *precision* 89,4%, *recall* 87,6%, dan *F-1 Score* 87,9%.

5.2 Saran

Penulis menyadari masih banyak kekurangan di dalam penelitian ini, dan masih banyak hal yang bisa untuk ditingkatkan lagi untuk penelitian ke depannya. Adapun saran penelitian untuk ke depannya ialah sebagai berikut:

1. Perlunya data uji yang lebih banyak lagi dan dalam satu tugas yang sama dari praktikan, yang di mana uji tersebut bisa mengukur lebih jauh lagi sistem yang telah dibuat.
2. Batas minimal baris *source code* yang bakal diuji diberi batasan minimal, karena jika baris *soure code* terlalu sedikit, maka kemungkinan sistem mendeteksi plagiat lebih besar.
3. Meluaskan bahan teliti ke bahasa pemrograman lain selain Java, yang di mana bahasa pemrograman juga selalu bertambah dan lebih variatif seiring berjalannya waktu.

DAFTAR PUSTAKA

Sutadi, *et al*. 2016. *Aplikasi Pendeteksi Kemiripan Isi Teks Dokumen Menggunakan Metode Levenshtein Distance*. semanTIK, Vol 2, No. 1. ISSN: 2502-8928.

Pratama dan Pamungkas. 2016. *Analisis Kinerja Algoritma Levenshtein Distance Dalam Mendeteksi Kemiripan Dokumen Teks*. Jurnal Logika, Jilid 6, No. 2. ISSN: 1978-8568.

Jimmy, *et al.* 2018. *Deteksi Kemiripan Source Code Dengan Metode Fingerprint Based Distance dan Levenshtein Distance.* Journal of Computer Science and Information Systems. ISSN: 2549-2829.

Ahmad dan Eka. 2014. *Deteksi Similiarity Source Code Menggunakan Metode Deteksi Abstract Syntax Tree*. Universitas Muhammadiyah Jakarta. ISSN: 2407-1846.

Yulianingsih. 2017. *Implementasi Algoritma Jaro-Winkler Dan Levenshtein Distance Dalam Pencarian Data Pada Database*. Jurnal String, Vol. 2, No. 1. ISSN: 2527-9661.

Rizqa dan Wiguna. 2014. *Pemanfaatan Algoritma Rabin-Karp Untuk Mengetahui Tingkat Kemiripan Dari Source Code Pemrograman Lisp*. Universitas Dian Nuswantoro. Semarang.

Febriyanto, *et al*. 2006. *Penerapan Algoritma Boyer-Moore untuk Pengecekan Plagiatisme Source Code.* Institut Teknologi Bandung.

Jimmy, *et al*. 2018. *Deteksi Kemiripan Source Code dengan Metode Fingerprint Based Distance dan Levenshtein Distance.* Journal of Computer Science and Information Systems, Vol. 2, No. 1.P-ISSN: 2549-2810.