IMPLEMENTASI ALGORITMA DAMERAU-LEVENSHTEIN DISTANCE PADA SISTEM PENDETEKSI PLAGIARISME KODE BERBASIS ABSTRACT SYNTAX TREE

**Abdullah Ammar1, Ahlijati Nuraminah2**

1,2Program Studi Ilmu Komputer

Sekolah Tinggi Ilmu Manajemen dan Ilmu Komputer ESQ  
Jl. TB Simatupang, RT.3/RW.3, Cilandak Timur., Jakarta 12560

Email: 1abdullah.a@students.esqbs.ac.id, 2ahlijati.nuraminah@esqbs.ac.id

(Naskah masuk: 18 Juli 2022, diterima untuk diterbitkan: dd mmm yyyy)

Abstrak

Plagiarisme kode merupakan fenomena yang sering terjadi pada pembelajaran pemrograman, begitu juga yang terjadi pada mata kuliah struktur data dan algoritma di prodi Ilmu Komputer STIMIK ESQ. Proses pengecekan berkas *source code* ini dilakukan secara manual dan membutuhkan waktu yang banyak hingga 1 sampai 2 jam untuk 20 berkas kode saja. Penelitian ini akan mengembangkan sistem pendeteksi plagiarisme kode berbasis *Abstract Syntax Tree* dengan menerapkan algoritma *Damerau-Levenshtein Distance* untuk mencari nilai similaritas antar berkas *source code* dan nilai F-Measure dari seluruh berkas *source code*. Dua pengujian yang dilakukan pada berkas *source code* *generated* dan mata kuliah struktur data dan algoritma tahun 2019-2021, membutuhkan waktu masing-masing sebesar 5.704 detik dan 0.996 detik. Dari pengujian tersebut juga didapatkan nilai F-Measure masing-masing sebesar 0.16 sebagai yang terendah dan tertinggi sebesar 0.8. Hasil tersebut menujukkan bahwa sistem mampu melakukan proses pendeteksian secara cukup cepat dan mampu mendeteksi secara efektif beberapa kecurangan umum yang ditemukan dalam plagiarisme kode namun tidak cukup efektif untuk mendeteksi kecurangan yang lebih rumit. Oleh karena itu peneliti memberikan rekomendasi untuk menambah langkah proses *preprocessing* untuk memangkas *node* yang tidak diperlukan dan mendeteksi kode yang setara namun berbeda sintaks.

**Kata kunci**: *code plagiarism, code similarity, abstract syntax tree, string distance, algoritma damerau-levensthein distance*

*IMPLEMENTATION OF DAMERAU-LEVENSHTEIN DISTANCE ALGORITHM IN ABSTRACT SYNTAX TREE-BASED CODE PLAGIARISM DETECTION SYSTEM*

*Abstract*

Code plagiarism is a phenomenon that often occurs in programming learning, as well as in data structures and algorithms courses in the STIMIK ESQ Computer Science study program. The process of checking the source code file is done manually and takes a lot of time, up to 1 to 2 hours for only 20 code files. This research will develop a code plagiarism detection system based on Abstract Syntax Tree by applying the Damerau-Levenshtein Distance algorithm to find the similarity value between source code files and F-Measure values from all source code files. Two tests carried out on source code generated files and data structure and algorithm courses in 2019–2021 took 5,704 seconds and 0.996 seconds, respectively. From this test, the F-Measure value was also obtained, each with 0.16 as the lowest and the highest of 0.8. These results state that the system can carry out the detection process fairly quickly and can detect effectively some common frauds found in code plagiarism but is not effective enough to detect more complex fraud. Thus, the researcher recommends adding preprocessing steps to trim unnecessary nodes and detect equal code but with different syntax.

**Keywords**: *code plagiarism, code similarity, abstract syntax tree, string distance, algoritma damerau-levensthein distance*

# PENDAHULUAN

Plagiarisme kode adalah menyalin kode yang ditulis oleh orang lain secara penuh maupun sebagian atau mencuri kode sumber orang lain (Cosma and Joy, 2008). Studi pada tahun 2017 menunjukkan bahwa 65% pelajar melakukan plagiarisme kode setidaknya sekali dalam masa studi mereka (Pierce and Zilles, 2017). Plagiarisme kode menjadi masalah karena melanggar nilai integritas yang dimiliki STIMIK ESQ dan juga masalah bagi para dosen karena para mahasiswa lebih tertarik untuk mendapatkan nilai daripada belajar dan mendapatkan pengalaman dalam pemrograman.

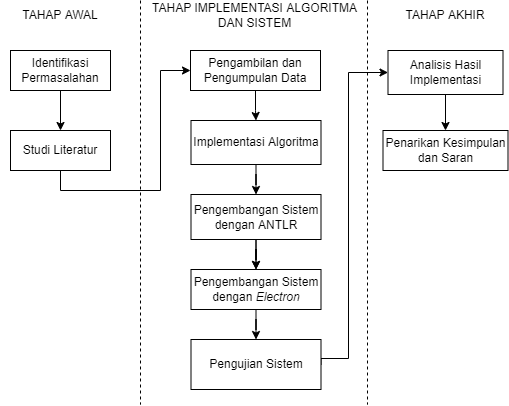
Pengalaman dalam pemrograman bisa didapatkan oleh pelajar melalui latihan ataupun mengerjakan tugas yang diberikan oleh dosen. Dalam mata kuliah struktur data dan algoritma di Prodi Ilmu Komputer STIMIK ESQ, setiap tugas dicek secara manual satu-persatu oleh dosen. Dosen memastikan bahwa tidak ada mahasiswa yang melakukan plagiarisme kode. Proses pengecekan secara manual ini membutuhkan banyak waktu dan sulit untuk dilakukan (Pandit and Toksha, 2019). Berdasarkan wawancara dengan dosen, waktu yang diperlukan untuk memproses kurang lebih 20 tugas adalah 1 hingga 2 jam lamanya. Dampak dari proses yang lama ini adalah sangat menguras tenaga dan menghabiskan waktu yang sebenarnya bisa dialihkan untuk kegiatan produktif yang lain. Oleh karena itu, untuk memangkas waktu proses dan mempermudah proses tersebut, dibutuhkanlah alat untuk mendeteksi plagiarisme kode.

Dalam mendeteksi plagiarisme kode, banyak cara deteksi dan algoritma yang bisa digunakan. Menurut Duracik dkk., (2018) *Structure-based Detection* yang menghasilkan Abstract Syntax Tree (AST) cocok digunakan untuk mendeteksi plagiarisme kode karena akurat dalam mempresentasikan sebuah struktur kode. AST juga dianggap efektif untuk mendeteksi kecurangan yang dilakukan untuk menghindari pengecekan sistem pendeteksi seperti penamaan ulang *variable*, penambahan komentar, dan penyusunan ulang fungsi (Nichols et al., 2019).

Untuk menghitung biaya yang dibutuhkan untuk mengubah kode satu ke kode yang lain dilakukan perhitungan menggunakan algoritma Damerau-Levenshtein Distance. Menurut Sri dkk., (2018) algoritma *Damerau-Levenshtein Distance* efektif digunakan untuk mendeteksi kesalahan eja. Menurut Rusdianto dan Chaniago (2018) algoritma ini juga efektif untuk digunakan dalam pendeteksian kode karena menghasilkan akurasi yang cukup tinggi berdasarkan *grammar rule* yang digunakan.

# METODOLOGI PENELITIAN

Pada Gambar 1**Error! Reference source not found.** merupakan representasi dari tiap-tiap tahapan yang dilakukan dalam pelaksanaan penelitian ini.



Gambar Alur Penelitian

Penelitian diawali dari tahap identifikasi permasalahan yang ada dalam cakupan ranah ilmu komputer, lalu dilakukan pendalaman terhadap studi literatur untuk memperkuat pemahaman teori pada topik yang diteliti.

Tahap selanjutnya adalah mengumpulkan data yang dibutuhkan untuk pengujian sistem. Data berasal dari dua sumber yang berbeda yaitu *program-generated data* yang dipublikasikan di GitHub dari penelitian terdahulu oleh Nichols dkk., (2019) dan data yang berasal dari mata kuliah Algoritma dan Struktur Data yang didapatkan melalui izin dari pihak Bagian Akademik STIMIK ESQ.

Proses selanjutnya adalah implementasi algoritma Damerau-Levenshtein Distance yang akan digunakan untuk menghitung biaya minimum yang dibutuhkan untuk mengubah baris kode yang satu ke yang lain. Setelah itu, dilakukan pengembangan sistem menggunakan ANTLR untuk mengubah kode sumber yang akan dideteksi kedalam sebuah Abstract Syntax Tree (AST). Hasil AST ini akan dimasukkan ke dalam proses *preprocessing* untuk dilakukan pemangkasan *node*, pengurutan fungsi dan variabel, dan penghapusan *output log*.

Setelah itu dilakukan pengujian sistem untuk mengukur tingkat akurasi sistem dalam mendeteksi plagiarisme kode menggunakan data yang telah didapatkan.

Tahap terakhir adalah penarikan simpulan yang diperoleh dari hasil pengujian. Simpulan berisi jawaban mengenai tingkat akurasi dan perbedaan waktu sebelum dan sesudah menggunakan sistem.

# HASIL DAN PEMBAHASAN

## Pengambilan dan Pengumpulan Data

Penelitian ini menggunakan data primer yang berasal dari bagian akademik STIMIK ESQ. Total data yang digunakan berjumlah 24 berkas *source code* yang diambil dari hasil *assesment* 2 Soal No.2 tahun ajaran 2019/2020 yang berjumlah 12 berkas *source code* dan dari hasil *assesment* 2 No.2 tahun ajaran 2020/2021 yang berjumlah 12 berkas *source code* atau setara dengan 66 pasang. Berdasarkan analisis secara manual, rata-rata berkas *source code* ini memiliki level kecurangan 4 dan baris kode sebanyak 100 hingga 300 baris kode. Rincian data dapat dilihat pada Tabel 1

Tabel Frekuensi Data Original dan Plagiat Mata Kuliah Struktur Data dan Algoritma Tahun 2019-2021

| No | Tahun | Data | Frekuensi |
| --- | --- | --- | --- |
| 1 | 2019/2020 | Original | 2020\_1.java  2020\_2.java  2020\_4.java  2020\_9.java  2020\_11.java |
| Prediksi Plagiat | 2020\_0.java  2020\_5.java 2020\_6.java  2020\_7.java  2020\_10.java  2020\_12.java  2020\_13.java |
| 2 | 2020/2021 | Original | 2021\_2.java  2021\_3.java  2021\_5.java  2021\_7.java  2021\_8.java |
| Prediksi Plagiat | 2021\_0.java 2021\_1.java  2021\_4.java  2021\_6.java 2021\_9.java  2021\_10.java  2021\_11.java |

Data sekunder menggunakan data yang berasal dari laman GitHub milik Nichols dkk. (2019) yang diunggah pada tahun 2019. Data merupakan berkas *source code* hasil *program-generated* yang dibagi menjadi dua kategori yaitu data asli dan data plagiat. Data original memiliki jumlah rata-rata baris kode sebanyak 141 baris kode dan data plagiat sebanyak 2063 baris. Keduanya memilki rata-rata level kecurangan 5. Rincian data dapat dilihat pada Tabel 2

Tabel Frekuensi Data Asli dan Plagiat

|  |  |  |
| --- | --- | --- |
| No | Data | Frekuensi |
| 1 | Original | 10 |
| 2 | Plagiat | 10 |

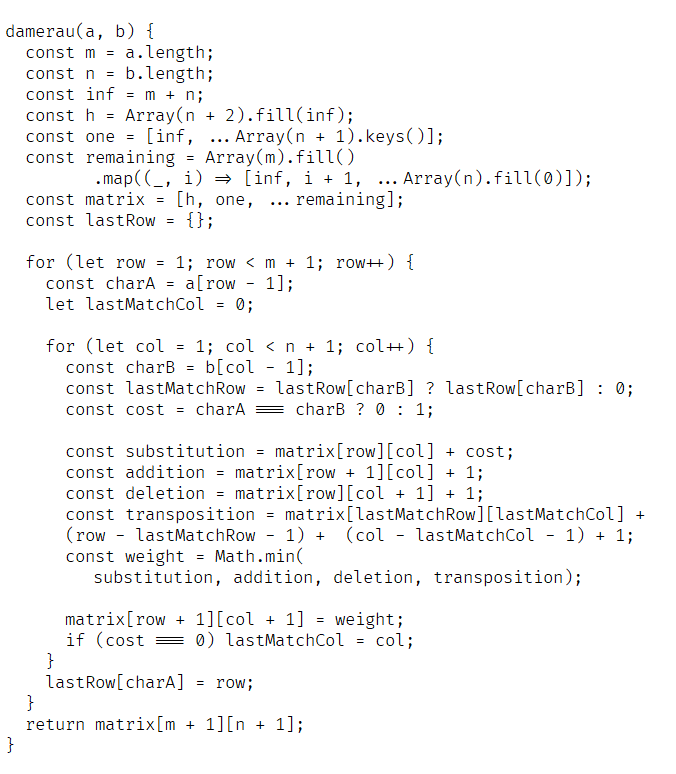
Adapun level kecurangan yang ada pada data di atas dan umum dalam plagiarisme kode menurut Wu dkk. (2019) yaitu:

1. Level 1: Salin dan tempel secara utuh
2. Level 2: Mengubah komentar
3. Level 3: Mengubah identifier (nama kelas, nama fungsi, dan nama variabel)
4. Level 4: Mengubah urutan kode.
5. Level 5: *Code refactoring* (pemisahan fungsi, penggantian kode yang setara)

## Implementasi Algoritma dan Pengembangan Sistem

Algoritma *Damerau-Levenshtein Distance* ini digunakan untuk menghitung biaya minimum untuk mengubah sebuah *string* S1 menjadi *string* S2. Algoritma ini merupakan pengembangan lebih lanjut dari algoritma *Levenshtein Distance* dan memiliki operasi yang sama yaitu operasi *insertion*, *deletion*, dan *substitution* dengan tambahan operasi *transposition* untuk menukar dua karakter yang berdekatan. Setiap biaya untuk operasi yang ada dihitung dan biaya yang paling minimum yang akan dipilih.

Algoritma ini diimplementasikan menggunakan bahasa pemrograman *JavaScript* berdasarkan penelitian oleh Damerau (1964) yang berjudul *A technique for computer detection and correction of spelling errors* dan juga penelitian oleh Wagner dan Lowrance (1975) yang berjudul *An Extension of the String-to-String Correction Problem*. Algoritma ini memiliki *time* dan *space complexity* pada skenario *worst case*. Algoritma Damerau-Levenshtein Distance dapat dilihat pada Gambar 2.



Gambar Algoritma Damerau-Levenshtein Distance

Setelah itu dilakukan pengembangan sistem dengan pembentukan Abstract Syntax Tree (AST) berdasarkan berkas *source code* menggunakan pustaka ANTLR4 (Another Tool for Language Recognition). Pustaka ini berguna untuk membuat *lexer* dan *parser* secara otomatis menggunakan aturan tata bahasa khusus yang disebut *grammar*. *Grammar* yang digunakan dalam penelitian ini merupakan grammar untuk bahasa pemrograman *Java* yang telah disediakan oleh ANTLR. *Grammar* ini nantinya akan digunakan oleh *lexer* dalam proses analisis leksikal sebagai aturan untuk mengubah urutan karakter ke dalam urutan token. Urutan token ini akan digunakan oleh *parser* untuk menghasilkan sebuah Concrete Syntax Tree (CST).

CST merupakan sebuah struktur *tree* yang menyimpan representasi penuh dari berkas source code. CST kemudian diubah ke dalam AST untuk menyederhanakan *tree* dan menyimpan informasi yang diperlukan saja. Adapun proses yang dilakukan untuk menyederhanakan struktur *tree* ini adalah seperti penghapusan *node* yang tidak diperlukan, penyederhanaan struktur *subtree*, dan pengurutan *subtree* untuk sebuah fungsi. Hasil dari proses ini nantinya akan masuk ke dalam proses linearisasi dimana struktur *tree* diubah menjadi struktur data larik yang berisikan *ruleIndex*-nya. Larik yang dihasilkan setelah proses linearisasi, akan digunakan untuk membandingkan kedua berkas source code menggunakan algoritma *Damerau-Levenshtein Distance*.

Setelah didapatkan biaya minimumnya, maka dilakukan perhitungan untuk mencari nilai similaritasnya menggunakan rumus *similarity* dan dilakukan pengurutan secara *ascending* dari nilai-nilai similaritas yang didapatkan untuk mempermudah proses menentukan mana berkas kode yang terindikasi melakukan plagiarisme paling tinggi. Rumus similarity yang digunakan adalah sebagai berikut (Rusdianto and Chaniago, 2018):

Pada persamaan (1), *m* dan *n* merupakan *string*, dimana *string* m akan diubah menjadi *string* n. *Distance[m,n]* merupakan biaya minimum untuk mengubah *string* *m* ke *n*. *Max(m,n)* adalah panjang *string* terpanjang antara *string m* dan *n*.

## Hasil Pengujian

Pengujian akurasi sistem dilakukan dengan memasukkan data yang telah dikumpulkan sebelumnya. Data pertama adalah *generated data* dari penelitian sebelumnya yang berjumlah 20 berkas *source code* dan yang kedua dari mata kuliah Struktur Data dan Algoritma yang berjumlah 24 berkas *source code*. Setiap data dikumpulkan ke dalam folder dan kemudian dimasukkan ke dalam sistem. Pengujian pertama untuk data hasil *generate* dengan *threshold* sebesar 0.5 dan pengujian kedua untuk data dari mata kuliah Struktur Data dan Algoritma dengan *threshold* 0.85

Pengujian pengukuran waktu dilakukan dengan membandingkan waktu yang dihabiskan oleh *expert* dalam melakukan proses deteksi plagiarisme kode secara manual dan waktu proses deteksi plagiarisme menggunakan sistem. Data yang digunakan berjumlah 7 berkas *source code* atau setara dengan 21 pasang berkas. Perhitungan waktu proses deteksi manual menggunakan *stopwatch* dan perhitungan waktu proses deteksi sistem menggunakan fungsi *perfomance.now()* bawaan *Node.js*.

Pengujian akurasi pertama yang dilakukan pada *generated* data sebanyak 20 berkas *source code* membutuhkan waktu sebanyak 5.704 detik. Dari hasil dari pengujian pertama, sistem memiliki akurasi yang sangat rendah dalam mendeteksi plagiarisme kode dengan skor *F-Measure* sebesar 0.16. Detail hasil pengujian akurasi pertama dapat dilihat pada Tabel 3.

Tabel Hasil Pengujian Akurasi Pertama

|  |  |
| --- | --- |
| Generated | |
| Threshold | 0.5 |
| Time | 5.704s |
| TP | 5 |
| FP | 5 |
| TN | 137 |
| FN | 43 |
| Precision | 0.5 |
| Recall | 0.1 |
| F-Measure | 0.16 |

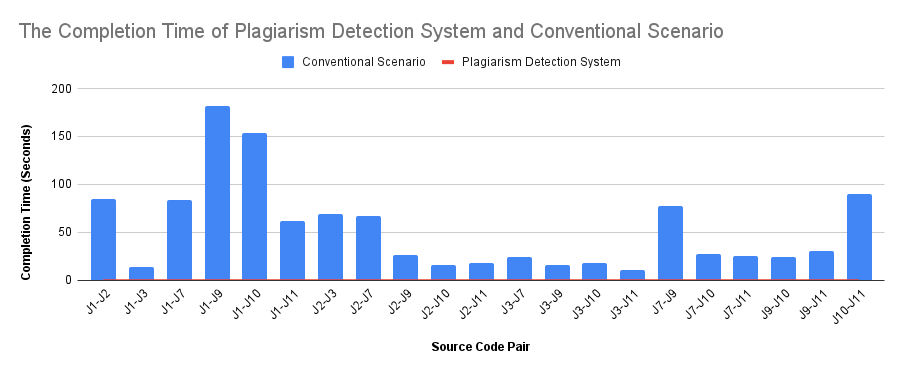
Kemudian, dilakukan pengujian kedua menggunakan data yang telah diolah dari mata kuliah Struktur Data dan Algoritma. Dalam pengujian kedua ini dilakukan pengujian sebanyak dua kali. Pertama untuk 12 berkas *source code assessment* 2 tahun 2019/2020 dan kedua untuk 12 berkas *source code assessment* 2 tahun 2020/2021. Pengujian pertama membutuhkan waktu sebanyak 0.996 detik dan pengujian kedua membutuhkan waktu sebanyak 0.631 detik. Detail hasil pengujian akurasi pertama dapat dilihat pada Tabel 4.

Tabel Hasil Pengujian Akurasi Kedua

|  |  |
| --- | --- |
| Tahun 2019/2020 | |
| Threshold | 0.85 |
| Time | 0.996s |
| TP | 2 |
| FP | 2 |
| TN | 60 |
| FN | 0 |
| Precision | 0.5 |
| Recall | 1 |
| F-Measure | 0.67 |
| Tahun 2020/2021 | |
| Threshold | 0.85 |
| Time | 0.631s |
| TP | 4 |
| FP | 2 |
| TN | 58 |
| FN | 0 |
| Precision | 0.67 |
| Recall | 1 |
| F-Measure | 0.8 |

Dari hasil kedua pengujian di atas, dapat dilihat bahwa pada pengujian akurasi pertama, sistem kurang mampu mendeteksi plagiarisme kode, ini dikarenakan berkas *source code* yang ada pada pengujian pertama memiliki berbagai kecurangan yang lebih beragam dan kompleks dibandingkan berkas *source code* dari pengujian kedua, sehingga sistem tidak mampu mendeteksi kecurangan tersebut secara efektif. Namun, pada pengujian kedua, dapat dilihat bahwa sistem mampu mendeteksi plagiarisme kode secara cukup efektif dikarenakan kecurangan yang ada berkas source code tidak terlalu rumit.

Kemudian, untuk hasil pengujian waktu pemrosesan pendeteksian plagiarisme kode secara manual dan pendeteksian plagiarisme kode secara otomatis menggunakan sistem menunjukkan perbedaan yang signifikan. Pemrosesan manual membutuhkan waktu rata-rata sebesar 54 detik atau 0.9 menit dan pemrosesan sistem membutuhkan waktu rata-rata sebesar 0.06 detik atau 0.001 menit untuk setiap pasang berkas *source code*. Sedangkan waktu total pemrosesan secara manual membutuhkan waktu sebesar 1140 detik atau 19 menit dan pemrosesan dengan sistem membutuhkan waktu sebesar 0.06 detik atau 0.01 menit..



Gambar Grafik Perbedaan Waktu Deteksi Plagiarisme Kode Secara Manual dan Otomatis

Perbedaan waktu sebelum dan sesudah menggunakan sistem secara keseluruhan dapat dilihat pada Gambar 3. Dapat dilihat bahwa grafik untuk waktu pengujian pendeteksian kode menggunakan sistem tidak terlihat karena membutuhkan waktu yang sangat sedikit, kurang dari 1 detik untuk setiap pasangan berkas *source code*.

## Rekomendasi Perbaikan

Berikut beberapa rekomendasi perbaikan berdasarkan hasil pengujian yang telah diperoleh:

1. Mengkombinasikan metode deteksi *syntax-based* dan *structure-based* yang bisa disebut *hybrid-based* atau menambahkan algoritma lain untuk meningkatkan akurasi sistem.
2. Menambahkan fungsi ke dalam sistem untuk mampu mendeteksi baris kode yang dianggap sebagai *template*.

# KESIMPULAN

Berdasarkan hasil dari analisis dan eksperimen yang telah dilakukan, dapat diperoleh simpulan sebagai yaitu penggunaan sistem otomatis mampu memangkas waktu yang dibutuhkan untuk memproses 7 berkas source code atau 21 pasang berkas source code dalam 1140 detik atau 19 menit hingga hanya membutuhkan waktu sebesar 0.6 detik atau 0.01 menit dalam melakukan proses pendeteksian plagiarisme kode pada mata kuliah Struktur Data dan Algoritma.

Hasil dari nilai F-Measure yang diperoleh dari pengujian deteksi sistem pada berkas source code yang dihasilkan secara generated dengan level kecurangan 5 adalah 0.16 yang menunjukkan bahwa sistem tidak mampu mendeteksi plagiarisme kode pada level kecurangan ke-5.

Hasil dari nilai F-Measure yang diperoleh dari pengujian deteksi sistem pada berkas source code mata kuliah struktur data dan algoritma tahun 2019-2021 dengan level kecurangan 4 adalah 0.8 dan 0.67 yang menujukkan bahwa sistem cukup efektif dalam mendeteksi plagiarisme kode pada level kecurangan ke-4.

# DAFTAR PUSTAKA

COSMA, G. and JOY, M., 2008. Towards a definition of source-code plagiarism. *IEEE Transactions on Education*, 51(2), pp.195–200. https://doi.org/10.1109/TE.2007.906776.

DAMERAU, F.J., 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3), pp.171–176. https://doi.org/10.1145/363958.363994.

DURACIK, M., KRSAK, E. and HRKUT, P., 2018. Scalable Source Code Plagiarism Detection Using Source Code Vectors Clustering. In: *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*. Beijing, China: IEEE.pp.499–502. https://doi.org/10.1109/ICSESS.2018.8663708.

NICHOLS, L., DEWEY, K., EMRE, M., CHEN, S. and HARDEKOPF, B., 2019. Syntax-based Improvements to Plagiarism Detectors and their Evaluations. In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: ACM.pp.555–561. https://doi.org/10.1145/3304221.3319789.

PANDIT, A.A. and TOKSHA, G., 2019. Review of Plagiarism Detection Technique in Source Code. In: *International Conference on Intelligent Computing and Smart Communication* . Singapore: Springer.pp.393–405. https://doi.org/10.1007/978-981-15-0633-8\_38.

PIERCE, J. and ZILLES, C., 2017. Investigating Student Plagiarism Patterns and Correlations to Grades. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM. https://doi.org/10.1145/3017680.3017797.

RUSDIANTO, S. and CHANIAGO, R., 2018. Penerapan Abstract Syntax Tree dan Algoritme Damerau-Levenshtein Distance untuk Mendeteksi  Plagiarisme pada Berkas Source Code. *Jurnal Telematika 13*, 13(2), pp.105–110.

SRI, C., ADITYA, K., RAHMAYANTI, V. and NASTITI, S., 2018. Document Plagiarism Detection Using Damerau Levenshtein Algorithm And Query Expansion. *Seminar Nasional Teknologi dan Rekayasa (SENTRA)*, pp.85–89.

WAGNER, R.A. and LOWRANCE, R., 1975. An Extension of the String-to-String Correction Problem. *Journal of the ACM*, 22(2), pp.177–183. https://doi.org/10.1145/321879.321880.