Pengembangan Sistem Koreksi Ejaan Menggunakan Modifikasi Levenshtein Distance Berbasis Kedekatan Tombol Keyboard

Alya Nur Rahmah - 13524081
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: alyanrrhmah@gmail.com , 13524081@std.stei.itb.ac.id

Abstract*—Kesalahan pengetikan (*typographical error* atau *typo*) merupakan masalah yang sering muncul pada proses penulisan teks digital. Salah satu metode yang umum digunakan untuk koreksi ejaan adalah *Levenshtein Distance*, yaitu algoritma yang mengukur tingkat perbedaan antara dua string berdasarkan operasi penyisipan, penghapusan, dan substitusi karakter. Pada makalah ini diusulkan modifikasi *Levenshtein Distance* dengan memberikan biaya substitusi berdasarkan kedekatan posisi tombol pada keyboard QWERTY. Jarak antar tombol dihitung menggunakan jarak Euclidean pada representasi koordinat dua dimensi. Sistem diimplementasikan menggunakan Java dan diuji pada 1.498 data sintetis yang terdiri atas tiga jenis kesalahan, yaitu *keyboard-proximity error*, *deletion*, dan *transposition*. Hasil pengujian menunjukkan bahwa metode yang diusulkan mampu meningkatkan akurasi koreksi pada kesalahan akibat salah menekan tombol yang berdekatan, dari 94,6% menjadi 98,8%. Akan tetapi, performa metode mengalami penurunan pada kasus *deletion* dan *transposition*. Analisis lebih lanjut menunjukkan bahwa biaya substitusi yang terlalu rendah menyebabkan algoritma cenderung memilih kandidat dengan panjang kata yang sama meskipun tidak sesuai dengan kata yang dimaksud. Hasil ini menunjukkan bahwa pembobotan berbasis posisi keyboard dapat meningkatkan akurasi pada jenis kesalahan tertentu, tetapi perlu dikombinasikan dengan mekanisme penyeimbang agar tidak menurunkan performa pada jenis kesalahan lainnya.

Keywords—Levenshtein Distance, koreksi ejaan, keyboard QWERTY, weighted edit distance, typographical error.

I. PENDAHULUAN

Perkembangan teknologi informasi telah membuat aktivitas penulisan teks melalui perangkat digital menjadi bagian yang tidak terpisahkan dari kehidupan sehari-hari. Berbagai bentuk komunikasi, mulai dari pesan singkat, surat elektronik, laporan akademik, hingga dokumen resmi, kini banyak dilakukan secara digital. Meskipun demikian, kesalahan pengetikan (typographical error atau typo) masih sering ditemukan. Kesalahan tersebut dapat terjadi karena kurangnya ketelitian pengguna, kecepatan mengetik yang tinggi, maupun kesalahan dalam menekan tombol keyboard. Dalam beberapa kasus, typo hanya menyebabkan gangguan kecil pada keterbacaan teks. Namun, pada dokumen yang menuntut ketelitian tinggi, kesalahan pengetikan dapat menimbulkan ambiguitas bahkan kesalahpahaman terhadap informasi yang $disampaikan$[3],[6]. Untuk mengatasi permasalahan tersebut, berbagai sistem koreksi ejaan otomatis (automatic spell correction) telah dikembangkan. Sistem semacam ini bertujuan untuk mendeteksi kata yang tidak sesuai dengan kamus dan memberikan saran kata yang dianggap paling tepat. Salah satu pendekatan yang banyak digunakan dalam sistem koreksi ejaan adalah pengukuran kemiripan antar kata menggunakan algoritma edit distance. Dengan pendekatan ini, kata yang salah dapat dibandingkan dengan kata-kata dalam kamus untuk mencari kandidat koreksi yang memiliki tingkat kemiripan tertinggi.

Algoritma Levenshtein Distance merupakan salah satu metode edit distance yang paling populer dan banyak digunakan dalam berbagai aplikasi pemrosesan $teks$[2],[4]. Algoritma ini menghitung jumlah minimum operasi penyuntingan yang diperlukan untuk mengubah suatu string menjadi string lain. Operasi yang diperhitungkan meliputi penyisipan (insertion), penghapusan (deletion), dan substitusi (substitution) karakter. Semakin kecil nilai jarak yang dihasilkan, semakin tinggi tingkat kemiripan antara kedua string tersebut. Kesederhanaan konsep dan kemudahan implementasi menjadikan algoritma ini banyak digunakan dalam sistem koreksi ejaan, pencarian kata, serta berbagai aplikasi pemrosesan bahasa alami lainnya. Meskipun efektif untuk mengukur kemiripan dua string, Levenshtein Distance standar memiliki keterbatasan karena memperlakukan seluruh operasi substitusi dengan biaya yang sama. Sebagai contoh, perubahan huruf 'a' menjadi 's' memiliki biaya yang identik dengan perubahan huruf 'a' menjadi 'p'. Padahal, dalam praktiknya kedua jenis kesalahan tersebut memiliki kemungkinan terjadinya yang berbeda. Pada keyboard QWERTY, tombol 'a' dan 's' terletak bersebelahan sehingga kesalahan penekanan lebih mungkin terjadi dibandingkan

kesalahan antara 'a' dan 'p' yang posisinya berjauhan. Akibatnya, algoritma standar belum sepenuhnya merepresentasikan pola kesalahan yang umum dilakukan pengguna saat $mengetik$[3].

Berdasarkan pengamatan tersebut, makalah ini mengusulkan modifikasi terhadap Levenshtein Distance dengan memanfaatkan informasi kedekatan posisi tombol pada keyboard QWERTY. Setiap tombol direpresentasikan sebagai koordinat pada bidang dua dimensi, kemudian jarak antar tombol dihitung menggunakan jarak Euclidean. Nilai jarak tersebut digunakan sebagai dasar dalam menentukan biaya substitusi karakter. Dengan pendekatan ini, substitusi antara huruf yang berdekatan pada keyboard akan memiliki biaya yang lebih kecil dibandingkan substitusi antara huruf yang berjauhan. Diharapkan pendekatan tersebut dapat meningkatkan kemampuan algoritma dalam menangani kesalahan pengetikan yang disebabkan oleh salah menekan tombol keyboard. Sebagai bentuk evaluasi, modifikasi yang diusulkan tidak hanya diuji pada jenis kesalahan yang secara langsung menjadi target pendekatan ini, tetapi juga pada beberapa jenis typo lain yang umum ditemukan, yaitu deletion dan transposition. Pengujian dilakukan menggunakan dataset sintetis yang dibangkitkan secara terkontrol sehingga perbandingan antara algoritma standar dan algoritma yang dimodifikasi dapat dilakukan secara objektif. Selain membandingkan tingkat akurasi, makalah ini juga menganalisis perilaku algoritma pada kasus-kasus ketika hasil koreksi yang diberikan tidak sesuai dengan kata yang diharapkan.

Melalui implementasi dan eksperimen yang dilakukan, makalah ini bertujuan untuk mengevaluasi efektivitas modifikasi Levenshtein Distance berbasis kedekatan tombol keyboard dalam sistem koreksi ejaan. Selain itu, makalah ini juga bertujuan untuk mengidentifikasi dampak yang muncul akibat perubahan fungsi biaya pada algoritma, baik dampak yang meningkatkan performa maupun dampak yang justru menurunkan akurasi pada jenis kesalahan tertentu. Dengan demikian, hasil yang diperoleh diharapkan dapat memberikan pemahaman yang lebih baik mengenai kelebihan dan keterbatasan pendekatan weighted edit distance dalam permasalahan koreksi ejaan.

II. DASAR TEORI

A. Pencocokan String (String Matching)

Pencocokan string (string matching) merupakan permasalahan dasar dalam ilmu komputer yang bertujuan untuk menemukan kemunculan suatu pola (pattern) di dalam sebuah $teks$[1],[2]. Dalam permasalahan ini diberikan sebuah teks T dengan panjang n karakter dan sebuah pola P dengan panjang m karakter. Tujuannya adalah menentukan apakah pola tersebut muncul di dalam teks serta menemukan posisi kemunculannya apabila ada.

Teknik pencocokan string banyak digunakan dalam berbagai aplikasi, seperti pencarian kata pada editor teks, mesin pencari web, analisis citra, serta $bioinformatika$[1],[2]. Berbagai algoritma telah dikembangkan untuk menyelesaikan permasalahan ini, mulai dari metode sederhana seperti brute force hingga metode yang lebih efisien seperti Knuth-Morris-Pratt (KMP) dan Boyer-Moore. Dalam konteks koreksi ejaan, konsep pencocokan string digunakan untuk membandingkan kata yang dimasukkan pengguna dengan kata-kata yang tersedia pada kamus sehingga dapat ditemukan kandidat kata yang paling sesuai.

B. Levenshtein Distance

Levenshtein Distance merupakan algoritma yang diperkenalkan oleh Vladimir Levenshtein pada tahun 1965 untuk mengukur tingkat kemiripan antara dua $string$[4]. Algoritma ini menghitung jumlah minimum operasi penyuntingan (*edit operations*) yang diperlukan untuk mengubah suatu string menjadi string lainnya. Semakin kecil jumlah operasi yang dibutuhkan, semakin tinggi tingkat kemiripan antara kedua string tersebut. Terdapat tiga jenis operasi dasar yang digunakan dalam perhitungan *Levenshtein Distance*, yaitu:

1. Insertion (*penyisipan*), yaitu menambahkan sebuah karakter ke dalam string.
2. Deletion (*penghapusan*), yaitu menghapus sebuah karakter dari string.
3. Substitution (*penggantian*), yaitu mengganti suatu karakter dengan karakter lain.

Sebagai contoh, untuk mengubah kata "makan" menjadi "makam" hanya diperlukan satu operasi substitusi, yaitu mengganti huruf 'n' menjadi 'm'. Oleh karena itu, nilai *Levenshtein Distance* antara kedua kata tersebut adalah 1.

Perhitungan *Levenshtein Distance* umumnya dilakukan menggunakan pendekatan *dynamic programming*. Misalkan diberikan dua string:

- String pertama $s$dengan panjang m
- String kedua $t$dengan panjang n

Dibangun sebuah matriks berukuran $(m+1) \times (n+1)$, di mana elemen $D(i,j)$menyatakan jarak minimum antara prefiks pertama sepanjang $i$karakter pada string $s$dan prefiks pertama sepanjang $j$karakter pada string t. Baris pertama dan kolom pertama diinisialisasi sebagai:

$$D(i,0) = i$$
$$D(0,j) = j$$

karena diperlukan sejumlah operasi penghapusan atau penyisipan untuk mengubah string menjadi string kosong. Selanjutnya, setiap elemen matriks dihitung menggunakan relasi rekursif:

$$D(i,j) = \min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+cost \end{cases}$$

dengan:

$$cost = \begin{cases} 0, & s_i = t_j \\ 1, & s_i \neq t_j \end{cases}$$

Nilai pada sel terakhir matriks, yaitu $D(m, n)$, merupakan nilai *Levenshtein Distance* antara kedua string. Keunggulan utama algoritma ini adalah kemampuannya untuk menangani berbagai jenis kesalahan pengetikan secara seragam. Oleh karena itu, *Levenshtein Distance* banyak digunakan dalam sistem koreksi ejaan, pencarian dokumen, bioinformatika, dan berbagai aplikasi *Natural Language Processing* (NLP).

C. *Weighted Levenshtein Distance*

Meskipun efektif dalam mengukur kemiripan dua string, *Levenshtein Distance* standar memiliki keterbatasan karena menganggap seluruh operasi penyuntingan memiliki biaya yang sama. Dalam implementasi dasar, biaya untuk operasi *insertion*, *deletion*, dan *substitution* biasanya bernilai 1, sedangkan pencocokan karakter yang sama memiliki biaya 0. Pendekatan tersebut tidak selalu mencerminkan kondisi nyata. Dalam proses pengetikan, beberapa jenis kesalahan memiliki kemungkinan terjadinya yang lebih tinggi dibandingkan kesalahan lainnya. Sebagai contoh, pengguna lebih mungkin salah menekan huruf 's' ketika ingin mengetik huruf 'a' dibandingkan salah menekan huruf 'p', karena posisi tombol 'a' dan 's' saling berdekatan pada keyboard.

Untuk mengatasi permasalahan tersebut dikembangkan konsep *Weighted Levenshtein Distance*. Pada metode ini, setiap operasi penyuntingan dapat diberikan bobot yang berbeda sesuai karakteristik kesalahan yang ingin $dimodelkan$[5]. Secara umum, persamaan rekursif *Weighted Levenshtein Distance* dapat dituliskan sebagai:

$$D(i,j) = \min \begin{cases} D(i-1,j) + w_d \\ D(i,j-1) + w_i \\ D(i-1,j-1) + w_s \end{cases}$$

dengan:

- w_d= biaya penghapusan (*deletion*)
- w_i= biaya penyisipan (*insertion*)
- w_s= biaya substitusi (*substitution*)

Berbeda dengan *Levenshtein Distance* standar yang menggunakan nilai konstan, biaya pada metode berbobot dapat berubah sesuai aturan tertentu.

D. *Kesalahan Pengetikan (Typographical Error)*

Kesalahan pengetikan atau *typographical error* (*typo*) merupakan kesalahan yang terjadi ketika proses memasukkan teks menggunakan perangkat input seperti $keyboard$[6]. Kesalahan ini dapat disebabkan oleh berbagai faktor, seperti kurangnya ketelitian pengguna, kecepatan mengetik yang tinggi, maupun posisi tombol yang saling berdekatan. Beberapa jenis typo yang umum ditemukan antara lain:

1. *Insertion*, yaitu munculnya karakter tambahan yang tidak seharusnya ada.
2. *Deletion*, yaitu hilangnya karakter dari sebuah kata.
3. *Substitution*, yaitu tergantinya suatu karakter dengan karakter lain.
4. *Transposition*, yaitu tertukarnya posisi dua karakter yang berdekatan.

Pemahaman terhadap jenis-jenis typo penting dalam pengembangan sistem koreksi ejaan karena setiap jenis kesalahan memiliki karakteristik yang berbeda.

E. *Keyboard QWERTY dan Jarak Euclidean*

Keyboard QWERTY merupakan tata letak keyboard yang paling banyak digunakan pada perangkat komputer modern. Pada tata letak ini, posisi setiap huruf dapat direpresentasikan sebagai koordinat pada bidang dua dimensi.

Untuk mengukur kedekatan antar tombol digunakan jarak Euclidean. Jika terdapat dua titik dengan koordinat (x_1, y_1) dan (x_2, y_2), maka jarak Euclidean didefinisikan sebagai:

$$d = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$$

Semakin kecil nilai jarak yang diperoleh, semakin dekat posisi kedua tombol pada keyboard. Dalam makalah ini, jarak Euclidean digunakan sebagai dasar untuk menentukan biaya substitusi karakter pada algoritma yang diusulkan. Dengan pendekatan tersebut, kesalahan pengetikan akibat penekanan tombol yang berdekatan dapat diberikan penalti yang lebih rendah dibandingkan kesalahan pada tombol yang berjauhan.

III. PERANCANGAN DAN IMPLEMENTASI

A. *Perancangan Sistem*

Sistem yang dikembangkan bertujuan untuk memberikan koreksi ejaan terhadap kata yang mengandung kesalahan pengetikan. Masukan sistem berupa sebuah kata yang diduga mengandung typo, sedangkan keluaran sistem berupa kandidat kata dari kamus yang memiliki tingkat kemiripan tertinggi terhadap kata masukan.

Secara umum, proses koreksi dilakukan dengan membandingkan kata masukan terhadap seluruh kata yang terdapat pada kamus. Untuk setiap pasangan kata, dihitung nilai jarak menggunakan dua pendekatan yang berbeda, yaitu Levenshtein Distance standar dan Levenshtein Distance yang dimodifikasi. Kandidat dengan nilai jarak terkecil dipilih sebagai hasil koreksi yang diberikan oleh sistem. Perbedaan utama antara kedua pendekatan tersebut terletak pada perhitungan biaya substitusi karakter. Pada Levenshtein Distance standar, seluruh operasi substitusi memiliki biaya yang sama, yaitu sebesar 1. Sementara itu, pada metode yang diusulkan, biaya substitusi ditentukan berdasarkan jarak fisik antar tombol pada keyboard QWERTY. Dengan pendekatan ini, kesalahan akibat penekanan tombol yang berdekatan diharapkan dapat dimodelkan secara lebih realistis.

Untuk mengevaluasi efektivitas modifikasi yang dilakukan, kedua algoritma diuji menggunakan dataset sintetis yang berisi berbagai jenis kesalahan pengetikan.

Hasil koreksi dari masing-masing algoritma kemudian dibandingkan dengan kata target yang sebenarnya untuk menghitung tingkat akurasi.

B. *Representasi Keyboard QWERTY*

Modifikasi yang diusulkan pada makalah ini memanfaatkan informasi posisi huruf pada keyboard QWERTY. Oleh karena itu, setiap huruf perlu direpresentasikan dalam bentuk koordinat sehingga hubungan spasial antar tombol dapat dihitung secara matematis.

Keyboard direpresentasikan sebagai tiga baris utama yang berisi huruf alfabet. Setiap huruf dipetakan ke pasangan koordinat berupa indeks baris dan indeks kolom. Sebagai contoh, huruf q direpresentasikan sebagai koordinat (0,0), huruf w sebagai (0,1), dan huruf a sebagai (1,0). Seluruh koordinat huruf disimpan dalam struktur data HashMap sehingga posisi suatu karakter dapat diperoleh dengan cepat ketika diperlukan dalam perhitungan biaya substitusi. Pendekatan ini dipilih karena sederhana dan cukup untuk merepresentasikan kedekatan relatif antar tombol pada keyboard. Dengan adanya representasi koordinat, jarak antar huruf dapat dihitung menggunakan jarak Euclidean yang kemudian digunakan dalam perhitungan Modified Levenshtein Distance. Berikut ini sebagian representasi koordinat yang digunakan ditunjukkan pada tabel berikut ini :

Huruf	Koordinat	Huruf	Koordinat
q	(0,0)	d	(1,2)
w	(0,1)	f	(1,3)
e	(0,2)	z	(2,0)
r	(0,3)	x	(2,1)
a	(1,0)	c	(2,2)
s	(1,1)	v	(2,3)

Tabel 3.2. Representasi koordinat pada keyboard

Pemetaan tersebut kemudian ditulis dalam kode pemrograman sebagai berikut ini :

```java
public class KeyboardMap {

    private static final Map<Character, int[]> MAP = new HashMap<>();
    private static final double MAX_DIST = Math.sqrt(2 * 2 + 9 * 9);

    static {
        char[] row0 = {'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p'};
        char[] row1 = {'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l'};
        char[] row2 = {'z', 'x', 'c', 'v', 'b', 'n', 'm'};
        for (int i = 0; i < row0.length; i++) MAP.put(row0[i], new int[]{0, i});
        for (int i = 0; i < row1.length; i++) MAP.put(row1[i], new int[]{1, i});
        for (int i = 0; i < row2.length; i++) MAP.put(row2[i], new int[]{2, i});
    }
```

Gambar 3.2. Representasi layout keyboard QWERTY

C. *Modifikasi Levenshtein Distance*

Pada algoritma standar, biaya operasi *insertion*, *deletion*, dan *substitution* diberikan nilai yang sama, yaitu sebesar 1. Pendekatan tersebut tidak mempertimbangkan karakteristik kesalahan yang terjadi saat pengguna mengetik. Dalam praktiknya, kesalahan pengetikan sering terjadi karena pengguna menekan tombol yang berdekatan dengan tombol yang seharusnya ditekan. Sebagai contoh, huruf *a* lebih mungkin tertukar dengan huruf *s* dibandingkan dengan huruf *p* karena posisi kedua tombol tersebut lebih dekat pada keyboard.

Untuk memodelkan kondisi tersebut, biaya substitusi ditentukan berdasarkan jarak Euclidean antar huruf pada keyboard. Jika terdapat dua karakter c_1 dan c_2 dengan koordinat (x_1, y_1) dan (x_2, y_2), maka jarak antar keduanya dihitung menggunakan Persamaan (1).

$$d(c_1, c_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (1)$$

Nilai jarak yang diperoleh kemudian dinormalisasi menggunakan jarak maksimum pada keyboard sehingga biaya substitusi berada pada rentang 0 hingga 1. Persamaan biaya substitusi yang digunakan ditunjukkan pada Persamaan (2).

$$cost(c_1, c_2) = \min\left(1, \frac{d(c_1, c_2)}{\sqrt{85}}\right) \qquad (2)$$

Apabila kedua karakter yang dibandingkan identik, biaya substitusi bernilai 0. Sebaliknya, jika kedua karakter berbeda, biaya substitusi akan bergantung pada jarak fisik antar tombol. Berikut ini implementasi biaya substitusinya :

```java
public class KeyboardMap {
    // Mengembalikan koordinat [baris, kolom] dari karakter c, atau null jika tidak dikenali
    public static int[] getPosition(char c) {
        return MAP.get(Character.toLowerCase(c));
    }

    // Menghitung jarak Euclidean antara dua tombol keyboard
    public static double euclideanDistance(char c1, char c2) {
        int[] p1 = getPosition(c1);
        int[] p2 = getPosition(c2);
        if (p1 == null || p2 == null) return MAX_DIST;
        double dr = p1[0] - p2[0];
        double dc = p1[1] - p2[1];
        return Math.sqrt(dr * dr + dc * dc);
    }

    // Menghitung biaya substitusi dinamis antara dua karakter
    public static double substitutionCost(char c1, char c2) {
        c1 = Character.toLowerCase(c1);
        c2 = Character.toLowerCase(c2);
        if (c1 == c2) return 0.0;
        return Math.min(1.0, euclideanDistance(c1, c2) / MAX_DIST);
    }
```

Gambar 3.3. Implementasi biaya substitusi

Sebagai contoh, huruf *a* dan *s* memiliki jarak yang relatif kecil sehingga menghasilkan biaya substitusi yang rendah. Sementara itu, pasangan huruf yang berjauhan pada keyboard akan menghasilkan biaya yang lebih besar. Dengan pendekatan ini, algoritma dapat memberikan penalti yang lebih rendah pada kesalahan yang memang umum terjadi saat mengetik.

D. *Implementasi Sistem Koreksi Ejaan*

Algoritma diimplementasikan menggunakan bahasa pemrograman Java. Program terdiri atas beberapa komponen utama, yaitu modul representasi keyboard, modul perhitungan jarak antar tombol, modul Levenshtein Distance standar, modul Levenshtein Distance yang dimodifikasi, modul koreksi ejaan, serta modul pengujian. Secara umum, sistem bekerja dengan membandingkan kata masukan terhadap seluruh kata yang terdapat dalam kamus. Untuk setiap kandidat kata, dihitung nilai jarak

menggunakan algoritma Levenshtein Distance standar maupun Modified Levenshtein Distance. Kandidat dengan nilai jarak terkecil dipilih sebagai hasil koreksi yang diberikan oleh sistem. Berikut ini struktur proyek yang digunakan

```
lya@Alya-Nur-Rahmah:~/Makalah-Stima$ tree
.
├── README.md
├── data
│   ├── dataset.csv
│   ├── hasil_eksperimen.csv
│   └── kamus.txt
├── run.sh
└── src
    └── main
        └── java
            └── spellchecker
                ├── DatasetGenerator.java
                ├── Experiment.java
                ├── KeyboardMap.java
                ├── LevenshteinModified.java
                ├── LevenshteinStandard.java
                └── SpellChecker.java

6 directories, 11 files
```

Gambar 3.4.1. Struktur proyek

Folder/File	Penjelasan
data/	Berisi dataset yang akan diujikan serta file output hasil pengujian
DatasetGenerator.java	Membangkitkan dataset sintetis yang digunakan pada proses pengujian
Experiment.java	Menjalankan eksperimen, menghitung akurasi, serta mengukur waktu eksekusi algoritma
KeyboardMap.java	Menyimpan representasi koordinat keyboard QWERTY dan menghitung biaya substitusi antar karakter
LevenshteinModified.java	Mengimplementasikan Modified Levenshtein Distance dengan biaya substitusi berbasis kedekatan tombol keyboard
LevenshteinStandard.java	Mengimplementasikan algoritma Levenshtein Distance standar sebagai metode pembanding (baseline)
SpellChecker.java	Melakukan pencarian kata terbaik dari kamus berdasarkan nilai jarak minimum

Tabel 3.4.1. Penjelasan komponen program

Modul KeyboardMap merupakan komponen utama yang mendukung modifikasi algoritma. Modul ini menyimpan posisi setiap huruf pada keyboard QWERTY dan menyediakan fungsi untuk menghitung jarak Euclidean antar tombol. Selain itu, modul ini juga menyediakan fungsi untuk menghitung biaya substitusi dinamis yang digunakan oleh Modified Levenshtein Distance. Sementara itu, LevenshteinStandard dan LevenshteinModified mengimplementasikan proses perhitungan jarak antar string menggunakan pendekatan dynamic programming. Kedua algoritma menggunakan matriks berukuran ((m+1)\times(n+1)), dengan (m) dan (n) menyatakan panjang kedua string yang dibandingkan. Perbedaan utama keduanya terletak pada perhitungan biaya substitusi.

Pada Levenshtein Distance standar, biaya substitusi selalu bernilai 1 ketika dua karakter berbeda.

```java
public class LevenshteinStandard {

    public static double distance(String s1, String s2) {
        int m = s1.length();
        int n = s2.length();
        double[][] dp = new double[m + 1][n + 1];

        for (int i = 0; i <= m; i++) dp[i][0] = i;
        for (int j = 0; j <= n; j++) dp[0][j] = j;

        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                double subCost = (s1.charAt(i - 1) == s2.charAt(j - 1)) ? 0.0 : 1.0;
                dp[i][j] = Math.min(
                    Math.min(dp[i - 1][j] + 1.0,   // deletion
                             dp[i][j - 1] + 1.0),  // insertion
                    dp[i - 1][j - 1] + subCost      // substitution
                );
            }
        }
        return dp[m][n];
    }
}
```

Gambar 3.4.2. Kode Standart Levenshtein Distance

Sebaliknya, pada Modified Levenshtein Distance biaya substitusi dihitung menggunakan fungsi yang disediakan oleh KeyboardMap.

```java
public class LevenshteinModified {

    public static double distance(String s1, String s2) {
        int m = s1.length();
        int n = s2.length();
        double[][] dp = new double[m + 1][n + 1];

        for (int i = 0; i <= m; i++) dp[i][0] = i;
        for (int j = 0; j <= n; j++) dp[0][j] = j;

        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                double subCost = KeyboardMap.substitutionCost(
                    s1.charAt(i - 1), s2.charAt(j - 1)
                );
                dp[i][j] = Math.min(
                    Math.min(dp[i - 1][j] + 1.0,   // deletion
                             dp[i][j - 1] + 1.0),  // insertion
                    dp[i - 1][j - 1] + subCost      // substitution (dinamis)
                );
            }
        }
        return dp[m][n];
    }
}
```

Gambar 3.4.3. Kode Modified Levenshtein Distance

Proses koreksi ejaan dilakukan oleh kelas SpellChecker. Kelas ini memuat seluruh kata dari kamus ke dalam memori, kemudian membandingkan kata masukan terhadap seluruh kata dalam kamus. Untuk setiap kandidat kata dihitung nilai jaraknya, kemudian kandidat dengan nilai jarak minimum dipilih sebagai hasil koreksi.

```java
public class SpellChecker {

    private final List<String> dictionary;

    public SpellChecker(String dictionaryPath) throws IOException {
        dictionary = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(dictionaryPath))) {
            String line;
            while ((line = br.readLine()) != null) {
                String word = line.trim().toLowerCase();
                if (!word.isEmpty()) dictionary.add(word);
            }
        }
        System.out.println("Kamus dimuat: " + dictionary.size() + " kata.");
    }

    // Mencari kata terdekat menggunakan SLD
    public String correctStandard(String typo) {
        typo = typo.toLowerCase();
        String best = "";
        double bestDist = Double.MAX_VALUE;
        for (String word : dictionary) {
            double d = LevenshteinStandard.distance(typo, word);
            if (d < bestDist) {
                bestDist = d;
                best = word;
            }
        }
        return best;
    }

    // Mencari kata terdekat menggunakan MLD
    public String correctModified(String typo) {
        typo = typo.toLowerCase();
        String best = "";
        double bestDist = Double.MAX_VALUE;
        for (String word : dictionary) {
            double d = LevenshteinModified.distance(typo, word);
            if (d < bestDist) {
                bestDist = d;
                best = word;
            }
        }
        return best;
    }

    public List<String> getDictionary() {
        return dictionary;
    }
}
```

Gambar 3.4.4. Kode kelas SpellChecker

Selain modul koreksi ejaan, sistem juga dilengkapi dengan DatasetGenerator yang digunakan untuk membangkitkan data uji secara otomatis. Dataset dibangun menggunakan tiga jenis kesalahan pengetikan yang umum terjadi, yaitu keyboard proximity, deletion, dan transposition. Data yang dihasilkan kemudian digunakan oleh kelas Experiment untuk membandingkan performa Standard Levenshtein Distance dan Modified Levenshtein Distance.

Secara garis besar, alur kerja sistem diawali dengan memuat kumpulan kata dari berkas kamus yang digunakan sebagai referensi dalam proses koreksi ejaan. Setelah itu, sistem membangkitkan dataset pengujian yang berisi berbagai bentuk kesalahan pengetikan. Setiap kata typo pada dataset kemudian dibandingkan dengan seluruh kata yang terdapat dalam kamus menggunakan algoritma *Standard Levenshtein Distance* maupun *Modified Levenshtein Distance*. Dari seluruh kandidat yang ada, sistem memilih kata dengan nilai jarak terkecil sebagai hasil koreksi. Hasil tersebut selanjutnya dibandingkan dengan kata yang seharusnya untuk menentukan apakah koreksi yang diberikan sudah benar atau belum. Berdasarkan hasil pengujian tersebut, sistem menghitung tingkat akurasi masing-masing algoritma sekaligus mengukur waktu eksekusi yang diperlukan selama proses pengujian.

E. Penyusunan Dataset Pengujian

Untuk mengevaluasi performa algoritma, dibuat dataset sintetis menggunakan kelas DatasetGenerator. Dataset dibangkitkan secara otomatis dari kumpulan kata yang terdapat dalam kamus sehingga setiap data uji memiliki kata target yang diketahui dengan pasti. Tiga kategori kesalahan pengetikan digunakan dalam penelitian ini, yaitu:

1. Keyboard Proximity, yaitu mengganti suatu karakter dengan karakter tetangganya pada keyboard QWERTY.
2. Deletion, yaitu menghapus satu karakter secara acak dari suatu kata.
3. Transposition, yaitu menukar posisi dua karakter yang bersebelahan.

Pada kategori *keyboard proximity*, sebuah karakter diganti dengan karakter lain yang berada di sekitar posisinya pada keyboard QWERTY. Kategori ini dirancang untuk mensimulasikan kesalahan akibat salah menekan tombol yang berdekatan. Pada kategori *deletion*, satu karakter pada kata dihapus secara acak sehingga menghasilkan kata dengan panjang yang lebih pendek dibandingkan kata aslinya. Sementara itu, kategori *transposition* dibuat dengan menukar posisi dua karakter yang bersebelahan sehingga urutan huruf pada kata menjadi berubah. Berikut adalah contoh dataset.csv :

data > dataset.csv

```csv
typo,kata_benar,kategori
mskan,makan,KEYBOARD_PROXIMITY
pwnulis,penulis,KEYBOARD_PROXIMITY
algorirma,algoritma,KEYBOARD_PROXIMITY
makn,makan,DELETION
penuls,penulis,DELETION
algortima,algoritma,DELETION
maakn,makan,TRANSPOSITION
penluis,penulis,TRANSPOSITION
algroitma,algoritma,TRANSPOSITION
```

Gambar 3.5. Cuplikan potongan dataset

Setiap kategori menghasilkan 500 sampel data uji sehingga total data yang dibangkitkan berjumlah sekitar 1500 sampel. Setelah proses validasi, diperoleh 1498 data yang digunakan dalam eksperimen.

IV. PENGUJIAN

A. Metode Pengujian

Pengujian dilakukan untuk membandingkan performa Standard Levenshtein Distance (SLD) dan Modified Levenshtein Distance (MLD) dalam melakukan koreksi ejaan. Dataset yang digunakan merupakan dataset sintetis yang dibangkitkan secara otomatis menggunakan kelas DatasetGenerator dan terdiri atas 1498 data uji. Dataset dibagi menjadi tiga kategori kesalahan pengetikan, yaitu keyboard proximity, deletion, dan transposition. Kategori keyboard proximity merepresentasikan kesalahan akibat salah menekan tombol yang berdekatan pada keyboard. Kategori deletion merepresentasikan kesalahan berupa

hilangnya satu karakter dari kata asli, sedangkan kategori transposition merepresentasikan kesalahan berupa pertukaran posisi dua karakter yang berdekatan.

Setiap data uji terdiri atas kata typo, kata target yang benar, serta label kategori kesalahan. Untuk setiap kata typo, sistem melakukan koreksi menggunakan SLD dan MLD secara terpisah. Hasil koreksi kemudian dibandingkan dengan kata target untuk menentukan apakah koreksi yang diberikan sudah benar. Selain tingkat akurasi, pengujian juga mengukur waktu eksekusi yang dibutuhkan oleh masing-masing algoritma. Pengukuran waktu dilakukan menggunakan fungsi System.nanoTime() pada Java dan dihitung untuk seluruh data uji.

B. Hasil Pengujian Akurasi

```
======================================================
        HASIL EKSPERIMEN: AKURASI (%)
======================================================
Kategori                  SLD (%)      MLD (%)      Selisih
------------------------------------------------------
KEYBOARD_PROXIMITY          94.6%        98.8%        +4.2%
DELETION                    84.6%        11.6%       -73.0%
TRANSPOSITION               62.4%        49.0%       -13.5%
------------------------------------------------------
RATA-RATA                   80.6%        53.1%       -27.4%
```

Gambar 4.2. Hasil pengujian akurasi

Berdasarkan hasil pengujian, MLD menunjukkan peningkatan akurasi pada kategori *keyboard proximity*. Akurasi meningkat dari 94,6% menjadi 98,8%, atau naik sebesar 4,2%. Hasil ini menunjukkan bahwa penggunaan kedekatan tombol keyboard sebagai dasar penentuan biaya substitusi mampu membantu sistem mengenali kesalahan yang memang disebabkan oleh salah penekanan tombol yang berdekatan. Namun, performa MLD mengalami penurunan yang cukup signifikan pada kategori *deletion* dan *transposition*. Pada kategori *deletion*, akurasi turun dari 84,6% menjadi 11,6%, sedangkan pada kategori *transposition* akurasi turun dari 62,4% menjadi 49,0%.

Akibat penurunan tersebut, rata-rata akurasi keseluruhan MLD hanya mencapai 53,1%, lebih rendah dibandingkan SLD yang mencapai 80,6%. Hasil ini menunjukkan bahwa modifikasi yang diusulkan memang efektif untuk menangani kesalahan berbasis kedekatan tombol keyboard, tetapi belum mampu menangani jenis kesalahan pengetikan lainnya secara baik.

C. Hasil Pengujian Waktu Eksekusi

```
======================================================
       HASIL EKSPERIMEN: WAKTU EKSEKUSI
======================================================
SLD  -- total:   251.44 ms | per kata: 0.1679 ms
MLD  -- total:   514.57 ms | per kata: 0.3435 ms
Overhead MLD: +104.6%
======================================================
```

Gambar 4.3. Hasil pengujian waktu eksekusi

Berdasarkan hasil pengujian, waktu eksekusi MLD lebih tinggi dibandingkan SLD. Total waktu yang dibutuhkan MLD mencapai 514,57 ms, sedangkan SLD hanya membutuhkan 251,44 ms. Dengan kata lain, implementasi MLD menghasilkan *overhead* sekitar 104,6% dibandingkan metode standar. Peningkatan waktu eksekusi ini terjadi karena setiap operasi substitusi pada MLD memerlukan perhitungan tambahan berupa pengambilan koordinat karakter, perhitungan jarak Euclidean, serta proses normalisasi biaya substitusi. Sebaliknya, pada SLD biaya substitusi cukup ditentukan menggunakan nilai konstan sehingga proses perhitungannya lebih sederhana.

Meskipun demikian, rata-rata waktu eksekusi kedua metode masih berada di bawah satu milidetik per kata sehingga keduanya tetap dapat digunakan untuk melakukan koreksi ejaan pada dataset berukuran kecil hingga menengah.

D. Analisis Hasil

Berdasarkan hasil pengujian, *Modified Levenshtein Distance* (MLD) menunjukkan peningkatan performa pada kategori *keyboard proximity*. Akurasi meningkat dari 94,6% menjadi 98,8%, yang menunjukkan bahwa penggunaan kedekatan posisi tombol keyboard berhasil membantu sistem mengenali kesalahan akibat salah menekan tombol yang berdekatan. Sebaliknya, pada kategori *deletion* dan *transposition*, performa MLD justru lebih rendah dibandingkan *Standard Levenshtein Distance* (SLD). Hal ini terjadi karena modifikasi yang dilakukan hanya memengaruhi biaya substitusi, sedangkan kesalahan pada kedua kategori tersebut lebih berkaitan dengan hilangnya karakter atau perubahan urutan karakter. Akibatnya, MLD sering memilih kandidat yang memiliki biaya substitusi rendah meskipun bukan kata yang benar.

Secara keseluruhan, hasil pengujian menunjukkan bahwa modifikasi yang diusulkan efektif untuk menangani kesalahan *keyboard proximity*, tetapi belum mampu memberikan peningkatan performa pada jenis kesalahan pengetikan lainnya. Meskipun demikian, hasil ini menunjukkan bahwa informasi posisi tombol keyboard dapat dimanfaatkan untuk meningkatkan kualitas koreksi ejaan pada kasus-kasus tertentu.

V. KESIMPULAN

Pada makalah ini telah diimplementasikan modifikasi algoritma Levenshtein Distance dengan memanfaatkan kedekatan posisi tombol pada keyboard QWERTY untuk menentukan biaya substitusi antar karakter. Modifikasi dilakukan dengan menghitung jarak Euclidean antar tombol sehingga kesalahan pengetikan akibat salah menekan tombol yang berdekatan dapat direpresentasikan dengan lebih baik dibandingkan pendekatan Levenshtein Distance standar.

Berdasarkan hasil pengujian terhadap 1498 data uji, Modified Levenshtein Distance (MLD) berhasil meningkatkan akurasi pada kategori keyboard proximity dari 94,6% menjadi 98,8%. Hasil ini menunjukkan bahwa informasi posisi tombol keyboard dapat dimanfaatkan untuk meningkatkan kemampuan sistem dalam menangani kesalahan yang disebabkan oleh kedekatan tombol. Namun, pada kategori deletion dan transposition, performa MLD masih berada di bawah Standard Levenshtein Distance (SLD), sehingga rata-rata akurasi keseluruhan MLD hanya mencapai 53,1%, lebih rendah dibandingkan SLD yang mencapai 80,6%.

Dari hasil tersebut dapat disimpulkan bahwa modifikasi yang diusulkan efektif untuk menangani kesalahan pengetikan yang berkaitan dengan posisi tombol keyboard, tetapi belum mampu memberikan peningkatan performa pada seluruh jenis kesalahan pengetikan. Oleh karena itu, pengembangan lebih lanjut masih diperlukan agar metode ini dapat mempertahankan keunggulannya pada kasus keyboard proximity tanpa mengurangi performa pada kategori kesalahan lainnya.

SOURCE CODE AT GITHUB

https://github.com/alyanrrhma/Makalah-Stima.git

Video Link at Youtube
https://bit.ly/ytbmakalahstima81

KATA PENUTUP

Penulis mengucapkan puji dan syukur ke hadirat Allah SWT atas segala rahmat, karunia, dan petunjuk-Nya sehingga makalah yang berjudul *" Pengembangan Sistem Koreksi Ejaan Menggunakan Modifikasi Levenshtein Distance Berbasis Kedekatan Tombol Keyboard"* ini dapat diselesaikan dengan baik. Penulis juga menyampaikan terima kasih kepada Bapak Dr. Rinaldi Munir, M.T., selaku dosen mata kuliah Strategi Algoritma Semester II Tahun Akademik 2025/2026, yang telah memberikan ilmu, arahan, serta wawasan yang menjadi landasan dalam penyusunan makalah ini. Penulis menyadari bahwa makalah ini masih memiliki berbagai keterbatasan dan jauh dari sempurna. Oleh karena itu, penulis terbuka terhadap kritik dan saran yang membangun untuk perbaikan di masa mendatang. Akhir kata, penulis memohon maaf apabila terdapat kesalahan maupun kekurangan dalam penyusunan makalah ini dan berharap makalah ini dapat memberikan manfaat bagi pembaca.

DAFTAR PUSTAKA

[1] R. Munir, "Pengantar Strategi Algoritma," Institut Teknologi Bandung, 2026. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/01-Pengantar-Strategi-Algoritma-(2026).pdf

[2] R. Munir, "Pencocokan String (String/Pattern Matching)," Institut Teknologi Bandung, 2026. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/24-String-Matching-dengan-Regex-(2026).pdf

[3] K. Kukich, "Techniques for Automatically Correcting Words in Text," ACM Computing Surveys, vol. 24, no. 4, pp. 377–439, 1992.

[4] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," Soviet Physics Doklady, vol. 10, no. 8, pp. 707–710, 1966.

[5] E. Brill dan R. C. Moore, "An Improved Error Model for Noisy Channel Spelling Correction," dalam Proc. 38th Annual Meeting ACL, Hong Kong, 2000, pp. 286–293.

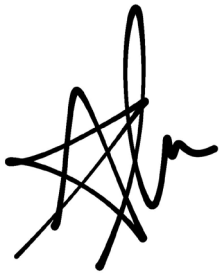
[6] A. I. Fahma, "Identifikasi Kesalahan Penulisan Kata (Typographical Error) pada Dokumen Berbahasa Indonesia Menggunakan Metode N-gram dan Levenshtein Distance," Jurnal Ilmiah Teknologi Informasi, 2018.

[7] P. Norvig, "How to Write a Spelling Corrector," 2016. [Online]. Tersedia: https://norvig.com/spell-correct.html

PERNYATAAN PRIBADI

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026



Alya Nur Rahmah 13524081