Validasi Pencarian Kata Kunci Menggunakan Algoritma Levenshtein Distance Berdasarkan Metode Approximate String Matching

1st *Nurul Fadhillah
Univeritas Muslim Indonesia
Fakultas Imu Komputer
Makassar, Indonesia
nurulfadhillah.fikom@gmail.com

2nd Huzain Azis
Univeritas Muslim Indonesia
Fakultas Imu Komputer
Makassar, Indonesia
huzain.azis@umi.ac.id

3rd Dirgahayu Lantara
Univeritas Muslim Indonesia
Fakultas Imu Komputer
Makassar, Indonesia
dirgahayu.lantara@umi.ac.id

***Abstrak*—Untuk mengatasi kesalahan dalam pencarian kata kunci perlu dilakukan optimasi proses pencarian pada aplikasi Kamus Besar Bahasa Indonesia (KBBI) digital. Namun, tidak sedikit ditemui kesalahan dalam menuliskan kata kunci sehingga menghasilkan keluaran yang tidak sesuai dengan keinginan pengguna. Dalam hal ini diperlukan sistem yang dapat melakukan koreksi hasil pencarian kata kunci pada aplikasi KBBI digital dalam bentuk validasi hasil pencarian. Penelitian ini menggunakan metode *Approximate String Matching* pada algoritma *Levenshtein Distance*. Pada metode ini, akan diketahui jarak *Levenshtein* yang menjadi nilai kemiripan suatu objek bertipe *string*. Untuk mendapatkan nilai kemiripan dilakukan dengan menghitung jarak antar dua *string* dengan menghitung jumlah operasi yang terjadi seperti penambahan, penghapusan atau pengurangan karakter. Semakin rendah nilai jarak antar dua *string* maka semakin tinggi tingkat kemiripan kedua *string* tersebut dan sebaliknya. Seperti pada tingkat kemiripan antara *string* "varitas" dengan *string* "varietas" memiliki tingkat kemiripan dengan melihat *Levenshtein Distance* sama dengan 1 karena hanya mengalami operasi 1 kali yaitu operasi penambahan karakter dan nilai akurasi similaritas sama dengan 88 %.**

Kata Kunci*— KBBI Digital, levenshtein distance, approximatte string matching, pencarian

I. PENDAHULUAN

Sistem pencarian objek bertipe *string* sudah banyak dikembangkan. Tetapi tidak sedikit fitur pencarian yang diimplementasikan dalam sistemnya adalah menampilkan hasil pencarian sesuai dengan kata kunci yang dimasukan. Ini artinya pengguna sistem ini harus betul-betul menambahkan kata kunci yang sudah dipastikan kebenarannya karena dengan penambahan yang salah akan menghasilkan *output* tidak sesuai dengan keinginan pengguna.

Sistem pencarian yang mulai dikembangkan adalah sistem pencarian yang sudah dapat memperkirakan kesalahan pengguna dalam menambahkan kata kunci. Menyelesaikan masalah bagaimana jika pengguna menambahkan kata kunci yang keliru atau tidak sesuai. Sistem yang dimaksud adalah sistem yang dapat mengoreksi kesalahan penambahan kata kunci pengguna.

Membuat koreksi pada penambahan kata kunci dilakukan dengan membandingkan *string* yang satu dengan lainnya. Pembandingan *string* secara garis besar dibedakan menjadi dua yaitu *exact string matching* (pembandingan *string* secara tepat dengan susunan karakter dalam *string* yang dibandingkan) dan *inexact string matching* (pembandingan *string* dimana *string* yang dicocokkan memiliki kemiripan namun keduanya memiliki susunan karakter yang berbeda) [1]. Untuk membuat koreksi objek *string* dapat digunakan metode *Approximate String Matching* dengan algoritma *Levenshtein Distance.* Algoritma ini digunakan untuk menentukan jarak *Levenshtein* antara dua objek yang bertipe *string*. Dua objek *string* yang dimaksud itu adalah *string* asal dan *string* target. Nilai jarak antar dua objek *string* ini akan diketahui dengan banyaknya jumlah operasi yang terjadi pada saat dilakukan perbandingan. Operasi yang dimaksud adalah operasi penambahan, pengurangan dan pengubahan karakter pada suatu *string*. Objek yang memiliki nilai jarak rendah artinya memiliki tingkat kemiripan yang tinggi sehingga objek dengan nilai jarak terendah akan menjadi saran untuk kata kunci baru.

Kelebihan dari metode *Approximate String Matching* adalah metode *Approximate String Matching* telah diuji dengan 5 algoritma berbeda yaitu *Levenshtein, Jaro Winkler, Monge Elkan, Bi-Gram* dan *Jaccard*, dan hasilnya *Approximate String Matching* memiliki tingkat akurasi, kinerja, pendekatan perkiraan, serta nilai MAP (*Mean Average Precission*) metrik yang cukup baik [2].

II. METODOLOGI

Dalam proses penggunaan kamus KBBI *digital*, pengguna akan menambahkan kata kunci untuk melakukan proses pencarian. Untuk melakukan proses pencarian ini kemungkinan terjadi kesalahan pada saat penambahan kata kunci. *String metric* adalah matriks berbasis karakter atau tekstual yang dapat menghasilkan nilai kesamaan atau ketidaksamaan dari dua buah teks *string* untuk proses perbandingan dan penyamaan. Beberapa metode yang menggunakan *string metric* seperti TF-IDF, *Smith-Waterman, Levenshtein Distance*, dll [3].

*Corresponding Author

A. Kamus Besar Bahasa Indonesia (KBBI)

Kamus adalah buku yang berisi rujukan yang menerangkan makna kata-kata. Kamus Besar Bahasa Indonesia (KBBI) adalah kamus ekabahasa resmi dari pusat bahasa yang berisi kumpulan kata-kata atau kosakata dalam bahasa Indonesia yang disusun berdasarkan alfabet. Didalam kamus memuat kosakata yang disertai dengan keterangan atau pemaknaannya. KBBI ini memiliki peran yang dianggap oleh penggunanya sebagai rujukan dan dianggap sebagai jalan keluar penyelesaian masalah tentang kata.

B. Validasi Pencarian

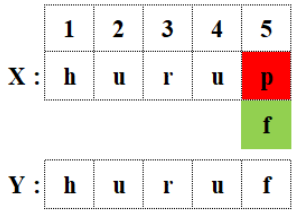
Proses pencarian adalah proses untuk mencari sebuah data tertentu dengan menggunakan kata kunci dalam pencariannya. Dalam proses pencarian akan diawali dengan memasukkan kata kunci kemudian data akan dicari didalam kamus data dengan membandingkan kata kunci yang dimasukkan dengan yang ada dalam kamus data. Dalam proses pencarian terutama dalam memasukkan kata kunci kemungkinan terjadi kesalahan sehingga hasil yang didapatkan tidak sesuai dengan keinginan pengguna. Atau hal ini dapat terjadi jika kata kunci yang dimasukkan oleh pengguna bukan merupakan kata kunci yang berupa kata baku. Validasi hasil pencarian ini merupakan proses pencocokan kata kunci jika pengguna keliru dalam memasukkan kata kunci. Validasi yang dimaksudkan dalam bentuk koreksi kata kunci yang dimasukkan dengan mencari dan membandingkan kata kunci dengan kata-kata lainnya dengan melihat similaritas antar kata.

C. Metode Approximate String Matching

Approximate String Matching yaitu teknik untuk membandingkan pola pada string dengan cara pendekatan [2]. Terdapat tiga operasi pada pendekatan tersebut yaitu :

1) Operasi Pengubahan Karakter

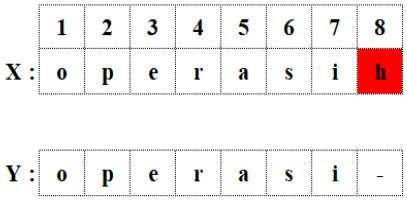
Operasi pengubahan karakter yaitu operasi penukaran suatu karakter dengan karakter lain contohnya pengguna menuliskan *string* "hurup" menjadi "huruf" dengan ilustrasi pada Gambar 1, dimana karakter pada indeks ke 5 pengubahan karakter ' p ' menjadi ' f '.



Gambar 1. Ilustrasi Operasi Pengubahan Karakter

2) Operasi Penghapusan Karakter

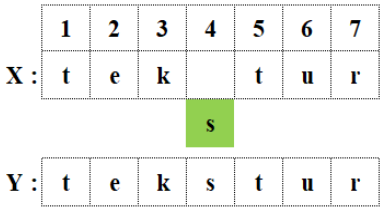
Operasi penghapusan karakter yaitu operasi menghapus suatu karakter dari suatu *string* contohnya *string* "operasih" menjadi "operasi" dengan ilustrasi pada Gambar 2, dimana karakter pada indeks ke 8 penghapusan karakter ' h '.



Gambar 2. Ilustrasi Operasi Penghapusan Karakter

3) Operasi Penambahan Karakter

Operasi penambahan karakter adalah operasi penambahan karakter terhadap suatu *string* contohnya *string* "tektur" menjadi "tekstur" dengan ilustrasi pada Gambar 3, dimana ditambahkan sebuah karakter "s" pada *string* "tektur". Penambahan *string* ini dapat dilakukan diawal, ditengah atau diakhir kata.



Gambar 3. Ilustrasi Operasi Penambahan Karakter

Approximate String Matching dapat diterapkan dalam berbagai algoritma, misalnya *Hamming, Levenshtein, Damerau Levenshtein, Jaro-Winkler, Wagner-Fisher,* dan lain-lain. *Approximate String Matching* adalah masalah komputasi yang terkenal dengan pencarian database, deteksi *plagiarism*, koreksi ejaan, dan bioinformatika [4].

D. Algoritma Levenshtein Distance

Algoritma *Levenshtein Distance* ditemukan oleh Vlamidir Levenshtein, seorang ilmuan asal Rusia pada tahun 1965, algoritma ini sering juga disebut dengan *Edit Distance* [5]. Algoritma ini digunakan secara luas dalam berbagai bidang, misalnya mesin pencari, pengecek ejaan (*spell checking*), pengenal pembicaraan (*speech recognition*), pengucapan dialeg, analisis DNA, pendeteksi pemalsuan, dan lain-lain [6].

Levenshtein Distance adalah suatu pengukuran untuk menghitung jumlah perbedaan antara dua kata. Perhitungan jarak antara dua kata ditentukan dari jumlah minimum operasi perubahan untuk mengubah kata A menjadi kata B [7]. Pada algortima *Levenshtein Distance*, pertama-tama sistem akan menfilter semua kata yang memiliki panjang antara (panjang kata kunci - 3) s.d (panjang kata kunci + 3). Pembatasan ini adalah batas aman yang ditetapkan dan dianggap sudah cukup untuk mendeteksi kesalahan ejaan yang dilakukan. Sementara nilai jarak antar 2 kata ini didapatkan dari jumlah operasi-operasi perubahan yang dilakukan antara satu kata dengan beberapa kata yang lain dengan bantuan matriks untuk

melakukan perubahan dari suatu *string* ke *string* lainnya. Total angka untuk tiap operasinya mengacu kepada *distance*, dimana semakin kecil *distance* semakin besar kemungkinan kata sesuai dengan target kata. Penempatan karakter pada array dalam bentuk matriks string ada pada tabel 1 [8].

TABEL 1. PENEMPATAN KARAKTER PADA ARRAY

		k	a	t	a	l
	0 (index 0)	1 (index 1)	2	3	4	5
k	1 (index 2)	? (index 3)				
a	2					
t	3					
a	4					
2	5					hasil

Index menunjukkan lokasi *cell* yang dibutuhkan untuk menghitung *distance*. Index ini mewakili index *pada two-dimensional array* yang terdiri dari *index* 0 (0,0), *index* 1 (0,1), *index* 2 (1,0) dan *index* 3 (1,1). *Index* 3 adalah *cell distance* untuk menghitung *distance* karakter. Keempat *cell* ini akan bergeser untuk memberi nilai *distance* pada masing-masing karakter [8], persamaan (1).

$$Bobot\ Similarity = \left(1 - \frac{d[m,n]}{Max(S,T)}\right) * 100\% \quad (1)$$

Setelah melakukan perhitungan untuk mendapatkan nilai *distance* dari dua *string* yang dibandingkan, selanjutnya menghitung nilai *similarity* dengan persamaan, yaitu [9] :

Dengan d[m, n] adalah nilai *distance*, terletak pada baris ke m dan kolom ke n, S adalah panjang *string* awal, T adalah panjang *string* target, dan *Max*(S, T) adalah panjang *string* terbesar antara *string* awal dan *string* akhir.

Bobot *similarity* diasumsikan pada rentang 0 hingga 100 persen, yang artinya nilai 100 adalah nilai maksimum yang menunjukkan bahwa kedua *string* tersebut sama identik. Pendekatan ini digunakan untuk mengukur bobot *similarity* antara dua *string* berdasarkan susunan karakter [9].

III. HASIL DAN PEMBAHASAN

Proses pencarian dengan menggunakan kata kunci "varitas", tidak menampilkan hasil karena kata kunci yang tidak sesuai. Hal ini dikarenakan sistem menganggap terjadi kesalahan pada kata kunci. Sistem dapat memberikan saran kata kunci "varietas". Proses yang dilakukan agar memperoleh kata kunci yang baru dilakukan dengan perbaikan kata kunci seperti sebagai berikut.

1) Konversi Kata Kunci

Proses konversi kata kunci dilakukan dengan memasukan kata kunci kedalam *array* misalnya kata "varitas".

2) Proses Seleksi Kata Kunci

Proses seleksi kata kunci "varitas" dengan semua kata yang ada pada tabel kata kunci. Kata kunci yang dapat dijadikan perbandingan adalah kata kunci dengan panjang karakter (P) antara $P_{katakunci}$-3 sampai $P_{katakunci}$+3. Karena "varitas" memiliki panjang 7 karakter, maka yang menjadi kata kunci pembanding mulai dari kata dengan panjang karakter 4–10 karakter. Misalnya, lima kata kunci yang dijadikan sebagai perbandingan yaitu kata "varian" dengan panjang 6 karakter, "varises" dengan panjang 7 karakter, "varietas" dengan panjang 8 karakter dan "variasi" panjang 7 karakter dan "fasilitas" dengan panjang 9 karakter .

3) Menghitung Jarak Levenshtein Terhadap Kata Kunci

Cara menghitung nilai jarak yaitu dengan membandingkan karakter yang berada pada posisi index yang sama. Perhitungan nilai jarak dimulai dari 0. Jika karakter yang dibandingkan sama maka nilai jarak tidak berubah. Jika karakter yang dibandingkan berbeda maka dilakukan operasi yang sesuai dan nilai jarak ditambah dengan 1. Untuk menghitung jarak *Levenshtein* menggunakan matriks seperti pada tabel 2 sampai Tabel VI.

a) Menghitung Jarak Kata Kunci dengan Kata "varian"

Berdasarkan representasi dari tabel 2, jarak antara "varitas" menjadi "varian" diperlukan 2 operasi, yaitu :

- Menghapus T

V A R I T A S → V A R I A S

- Mengubah S dengan N

V A R I A S → V A R I A N

TABEL 2. MATRIKS MENGHITUNG JARAK KATA KUNCI DENGAN KATA VARIAN.

			Kata Kunci Asal						
			V	A	R	I	T	A	S
		0	1	2	3	4	5	6	7
Kata Kunci	V	1	0	1	2	3	4	5	6
	A	2	1	0	1	2	3	4	5
	R	3	2	1	0	1	2	3	4
	I	4	3	2	1	0	1	2	3
Target	A	5	4	3	2	1	1	1	2
	N	7	5	4	3	2	2	2	2

b) Menghitung Jarak Kata Kunci dengan Kata "varises"

Berdasarkan representasi dari tabel 3, jarak antara "varitas" dan "varises" yaitu 2, berarti untuk mengubah kata "varitas" menjadi "varises" diperlukan 2 operasi, yaitu :

- Mengubah T dengan S

V A R I T A S → V A R I S A S

- Mengubah A dengan E

V A R I S A S → V A R I S E S

TABEL 3. MATRIKS MENGHITUNG JARAK KATA KUNCI DENGAN KATA VARISES.

			Kata Kunci Asal						
			V	A	R	I	T	A	S
		0	1	2	3	4	5	6	7
Kata Kunci Target	V	1	0	1	2	3	4	5	6
	A	2	1	0	1	2	3	4	5
	R	3	2	1	0	1	2	3	4
	I	4	3	2	1	0	1	2	3
	S	5	4	3	2	1	1	2	3
	E	6	5	4	3	2	2	2	3
	S	7	6	5	4	3	3	3	2

TABEL 4. MATRIKS MENGHITUNG JARAK KATA KUNCI DENGAN KATA VARIETAS.

			Kata Kunci Asal						
			V	A	R	I	T	A	S
		0	1	2	3	4	5	6	7
Kata Kunci Target	V	1	0	1	2	3	4	5	6
	A	2	1	0	1	2	3	4	5
	R	3	2	1	0	1	2	3	4
	I	4	3	2	1	0	1	2	3
	E	5	4	3	2	1	1	2	3
	T	6	5	4	3	2	1	2	3
	A	7	6	5	4	3	2	1	2
	S	8	7	6	5	4	3	2	1

c) Menghitung Jarak Kata Kunci dengan Kata "varietas"

Berdasarkan representasi dari tabel 4, jarak antara "varitas" dan "varietas" yaitu 1, berarti untuk mengubah kata "varitas" menjadi "varietas" diperlukan 1 operasi, yaitu:

- Menambahkan E

V A R I T A S → V A R I E T A S

d) Menghitung Jarak Kata Kunci dengan Kata "variasi"

Berdasarkan representasi dari tabel 5, jarak antara "varitas" dan "variasi" yaitu 3, berarti untuk mengubah kata "varitas" menjadi "variasi" diperlukan 3 operasi, yaitu :

- Mengubah T dengan A

V A R I T A S → V A R I A A S

- Mengubah A dengan S

V A R I A A S → V A R I A S A

- Mengubah A dengan I

V A R I A S A → V A R I A S I

TABEL 5. MATRIKS MENGHITUNG JARAK KATA KUNCI DENGAN KATA VARIASI.

			Kata Kunci Asal						
			V	A	R	I	T	A	S
		0	1	2	3	4	5	6	7
Kata Kunci Target	V	1	0	1	2	3	4	5	6
	A	2	1	0	1	2	3	4	5
	R	3	2	2	0	1	2	3	4
	I	4	3	3	1	0	1	2	3
	A	5	4	4	2	1	1	2	3
	S	6	5	5	3	2	2	2	3
	I	7	6	6	4	3	3	3	3

e) Menghitung Jarak Kata Kunci dengan Kata "fasilitas"

Sedangkan representasi dari tabel 6, jarak antara "varitas" dan "fasilitas" yaitu 4, berarti untuk mengubah kata "varitas" menjadi "fasilitas" diperlukan 4 operasi, yaitu:

- Mengubah V dengan F

V A R I T A S → F A R I T A S

- Mengubah R dengan S

F A R I T A S → F A S I T A S

- Menambahkan L

F A S I T A S → F A S I L T A S

- Menambahkan I

F A S I L T A S → F A S I L I T A S

TABEL 6. MATRIKS MENGHITUNG JARAK KATA KUNCI DENGAN KATA FASILITAS.

			Kata Kunci Asal						
			V	A	R	I	T	A	S
		0	1	2	3	4	5	6	7
Kata Kunci Target	F	1	1	2	3	4	5	6	7
	A	2	2	1	2	3	4	5	6
	S	3	3	2	2	3	4	5	6
	I	4	4	3	3	2	3	4	5
	L	5	5	4	4	3	3	4	5
	I	6	6	5	5	4	4	5	6
	T	7	7	6	6	5	4	6	7
	A	8	8	7	7	6	6	4	6
	S	9	9	8	8	7	7	6	4

Dari proses perhitungan pada tabel 2, tabel 3, tabel 4, tabel 5 dan tabel 6 didapatkan nilai jarak untuk masing-masing kata yang dibandingkan yaitu sebagai berikut.

Nilai jarak (varitas, varian) = 2

Nilai jarak (varitas, varises) = 2

Nilai jarak (varitas, varietas) = 1

Nilai jarak (varitas, variasi) = 3

Nilai jarak (varitas, fasilitas) = 4

4) Membandingkan Nilai Jarak Levenshtein

Berdasarkan nilai jarak, urutkan setiap kata yang dibandingkan mulai dari nilai jarak terendah sebagai kata kunci saran teratas karena kata dengan nilai jarak terendah memiliki kesamaan yang paling tinggi. Sehingga pada kasus ini kata "varietas" terpilih sebagai kata saran bagi kata "varitas".

5) Menghitung Nilai Similarity

Untuk menghitung similaritas dari dua string digunakan rumus similarity dengan menghitung jarak *levenshtein* dan panjang string terbesar yang dibandingkan. Untuk nilai similaritas untuk kasus ini digunakan persamaan 1. Similaritas antara "varitas" dengan "varian" dimana nilai jaraknya (d[m,n]) = 2, panjang (P) "varitas" = 7 dan panjang (P) "varian" = 6 yaitu :

$$similarity = \left(1 - \frac{d\,[m,n]}{Max\,(S,T)}\right) * 100\%$$

$$similarity = \left(1 - \frac{2}{Max\,(7,\ 6)}\right) * 100\%$$

$$similarity = \left(1 - \frac{2}{7}\right) * 100\%$$

$$similarity = (1 - \ 0{,}29\ ) * 100\%$$

$$similarity = (\ 0{,}71\ ) * 100\%$$

$$similarity = 71\ \%$$

Dengan perhitungan tersebut, hasil nilai *similarity* dari 5 *string* diatas ada pada tabel 7. Jika diketahui nilai S = 7 adalah nilai panjang (P) dari kata "varitas", dan T adalah nilai panjang dari kata.

TABEL 7. HASIL PERHITUNGAN SIMILARITY

No	Kata	T	d[m,n]	Max(S,T)	Similarity
1	varian	6	2	7	71%
2	varises	7	2	7	71%
3	varietas	8	1	8	88%
4	variasi	7	3	7	57%
5	fasilitas	9	4	9	56%

String dengan nilai similaritas tertinggi itu didapatkan dari nilai jarak yang rendah. Untuk kasus ini kata saran yang memiliki nilai jarak terendah dan nilai similaritas tertinggi yaitu "varietas".

IV. KESIMPULAN

Berdasarkan hasil penelitian ini, Algoritma *Levenshtein Distance* dapat digunakan untuk mengoreksi penambahan kata kunci yang keliru dengan cara membandingkan beberapa kata kunci berdasarkan panjang kata kunci. Koreksi dilakukan dengan pengecekan kesalahan terhadap tiap-tiap karakter dalam suatu *string*. Seperti kata "varitas" didapatkan hasil kata "varietas" sebagai kata yang disarankan mempunyai nilai jarak *levenshtein* sama dengan 1 dengan nilai similaritas 88%. Semakin panjang kata kunci maka akan membutuhkan waktu yang lebih lama. Perhitungan untuk mengukur nilai similaritas dapat dilakukan dengan mengetahui nilai jarak antar dua string. Tetapi untuk mendapatkan nilai akurasi maka perlu dilakukan perhitungan nilai similarity untuk mendapatkan nilai similaritas tertinggi.

DAFTAR PUSTAKA

[1] Y. Rochmawati and R. Kusumaningrum, "Studi Perbandingan Algoritma Pencarian String dalam Metode Approximate String Matching untuk Identifikasi Kesalahan Pengetikan Teks," vol. 7, pp. 125–134, 2016.

[2] M. O. Braddley and M. Fachrurrozi, "Koreksi Ejaan Kata Berbahasa Indonesia Menggunakan Algoritma Levensthein Distance," vol. 3, no. 1, pp. 167–171, 2017.

[3] I. Bagus and K. Surya, "Implementasi Algoritma Levenshtein Pada Sistem Pencarian Judul Skripsi / Tugas Akhir," vol. 11, pp. 46–53, 2017.

[4] M. Rubio, A. Alba, M. Mendez, and E. Arce-santana, "2013 Iberoamerican Conference on Electronics Engineering and Computer Science A Consensus Algorithm for Approximate String Matching," J. Mater. Process. Tech., vol. 7, pp. 322–327, 2013.

[5] R. B. Aplikasi, Z. Afriansyah, and D. Puspitaningrum, "Menggunakan Algoritma Levenshtein Distance (Studi Kasus : DNA Kanker Hati Manusia)," vol. 3, no. 2, pp. 61–67, 2015.

[6] A. S. Dewi Rokhmah Pyriana , Suprapto, "Program Aplikasi Editor Kata Bahasa Indonesia Menggunakan Metode Approximate String Matching Dengan Algoritma Levenshtein Distance Berbasis Java," pp. 1–10, 2012.

[7] M. Ben, "Adaptating the levenshtein distance to contextual spelling correction," vol. 12, no. 1, pp. 127–133, 2015.

[8] J. Informatika, F. Matematika, and P. Alam, "Menggunakan Levenshtein Distance Pada Layout Qwerty," no. Selisik, pp. 171–176, 2016.

[9] B. P. Pratama, "Analisis Kinerja Algoritma Levenshtein Distance," no. 2, pp. 131–143, 2016.