Jurnal Ilmiah Wahana Pendidikan, Mei 2025, 11(5.D), 89-104
DOI: https://jurnal.peneliti.net/index.php/JIWP/article/view/11944
p-ISSN: 2622-8327 e-ISSN: 2089-5364
Accredited by Directorate General of Strengthening for Research and Development
Available online at https://jurnal.peneliti.net/index.php/JIWP

Optimalisasi Proses Pencarian Data *Letter-C* Melalui Penerapan Algoritma *Levenshtein Distance* Di Desa Pandanarum

Celvinanda Octiawan[1], Indyah Hartami Santi[2], Filda Febrinita[3]

Fakultas Teknologi Informasi, Universitas Islam Balitar[1,2,3]

Received: 20 April 2025
Revised: 27 April 2025
Accepted: 01 Mei 2025

Abstract

Letter C adalah dokumen pendaftaran tanah penting di Desa Pandanarum. Namun, dokumentasi kepemilikan tanah yang buruk menyebabkan proses pencarian data *Letter C* menjadi lambat. Penelitian ini bertujuan mengoptimalkan pencarian data *Letter C* menggunakan algoritma *Levenshtein Distance.* Metode deskriptif kuantitatif digunakan dengan objek penelitian 464 data *Letter C* di Desa Pandanarum. Hasil menunjukkan bahwa algoritma *Levenshtein Distance* meningkatkan kecepatan dan akurasi pencarian data. Waktu pencarian berkurang hingga 60% dibandingkan metode manual, dan akurasi pencarian meningkat sebesar 40%, memungkinkan identifikasi dan koreksi kesalahan pengetikan. Kesimpulannya, algoritma *Levenshtein Distance* efektif mengoptimalkan pencarian data *Letter C,* menawarkan solusi efisien untuk masalah pencarian data tanah di Desa Pandanarum dan desa lainnya dengan masalah serupa. Implementasi algoritma ini tidak hanya menghemat waktu tetapi juga meningkatkan keakuratan data penting dalam administrasi pertanahan.

Keywords: *Letter C,* pencarian data, *Levenshtein Distance*, administrasi pertanahan, optimasi data tanah, Desa Pandanarum

(*) Corresponding Author: celvinanda.octiawan.22@gmail.com, indyahartamisanti@gmail.com, febrinitafilda80@gmail.com

How to Cite: Hartati, D., Darmiyanti, A., & Riana, N. (2025). Pengaruh Penggunaan Media Smart Board Alfabet Terhadap Kemampuan Keaksaraan Anak Usia Dini 5-6 Tahun Di Paud Sps Dahlia Serang Baru. *Jurnal Ilmiah Wahana Pendidikan*, *11*(5.D), 76-88. Retrieved from https://jurnal.peneliti.net/index.php/JIWP/article/view/11944

PENDAHULUAN

Dalam dinamika reformasi agraria pada Desa Pandanarum, yang terletak di selatan Kabupaten Blitar, menjadi representasi nyata dalam menghadapi tantangan kepemilikan tanah. Desa Pandanarum, dengan luas wilayah 369 hektar, 4 dusun, dan 7 RW, rumah bagi 8.300 jiwa dari 2.561 kepala keluarga yang beraktivitas di dusun, persawahan, dan pekarangan, yang berada pada ketinggian rata-rata 170 meter di atas permukaan laut. Kondisi ini menjadi tantangan besar dalam kepemilikan tanah yang masih krusial, seperti yang ditunjukkan oleh penelitian Santi, dkk. (2023). Dengan kondisi geografis ini, pemerintah berusaha untuk menyelesaikan tantangan dalam kepemilikan tanah yang kurang terdokumentasi dengan baik dan kurangnya pemahaman tentang sejarah tanah sehingga sebagian besar penduduk tidak memiliki akta tanah yang menyebabkan perselisihan kepemilikan karena batas-batas yang kurang jelas antara tanah yang saling berdekatan dan bahkan memiliki sertifikat ganda untuk satu bidang tanah (Soepandi & Widodo, 2021).

Selain itu, pengelolaan dan pencatatan data kepemilikan tanah mengalami kendala signifikan, terutama berkaitan dengan buku *Letter-C.* Buku ini, yang berperan sebagai dokumen pendaftaran tanah secara turun-temurun (Setiawan dkk., 2022), pada saat ini ada tiga buku dengan total 3.916 lembar data kepemilikan tanah yang terakhir dihitung

pada 1 Februari 2024. Dengan kondisi data yang kurang terstruktur dan adanya data yang kurang sesuai dengan data resmi dari Dinas Kependudukan dan Pencatatan Sipil, menimbulkan kesulitan dalam menjaga integritas informasi. Permasalahan ini diperparah oleh kerusakan fisik akibat usia kertas yang sangat tua, menandai perlunya perawatan khusus untuk setiap lembaran agar kerusakan dapat diminimalisir.

Permasalahan ini menggarisbawahi perlunya solusi untuk menghadapi tantangan serius dalam proses pencarian buku *Letter-C* secara manual. Hal ini menuntut pendekatan yang lebih inovatif dalam proses pencarian dengan waktu singkat. Melalui penerapan teknologi untuk mendaftarkan tanah dari buku menjadi bentuk digital, proses pencarian dapat dilakukan. Namun, mengingat bahwa data *Letter-C* tersebut tidak terstruktur dan kurang sesuai dengan data yang ada pada Dinas Kependudukan dan Pencatatan Sipil, dimana informasi ini mungkin hanya mudah dipahami oleh orang yang menulisnya, sedangkan mesin kesulitan untuk memproses dan menampilkan hasil yang diharapkan (Arifin & Prasetyo, 2021).

Dalam konteks pencarian dan pengelolaan data kepemilikan tanah di Desa Pandanarum, pendekatan tanpa algoritma sering kali menemui keterbatasan, terutama ketika data yang dicari dan data yang tersimpan tidak sepenuhnya sama. Proses manual dan sederhana ini sering kali gagal mengakomodasi varian penulisan atau kesalahan kecil yang umum terjadi dalam data tidak terstruktur. Misalnya, pencarian nama "Celvinanda" tidak akan berhasil jika inputnya adalah "calvinanda" menggunakan metode pencocokan string sederhana tanpa algoritma, menggaris bawahi perlunya pendekatan yang lebih canggih (Arifin & Prasetyo, 2021). Sehingga, penting untuk memastikan bahwa pengguna mendapatkan hasil yang relevan dengan kata kunci yang dimasukkan. Pada teori *String Matching*, seperti yang dipelajari dari karya Equi dkk., (2023) yang mengimplementasikan pencocokan string yang diterapkan dalam mesin pencari. Salah satu pendekatan sederhana yang umum digunakan dalam proses ini adalah metode Brute Force, di mana menjadi kelemahan dalam mencocokkan pola pencarian seperti yang telah diterapkan oleh Sinaga, (2021) pada proses pencariannya yang harus benar-benar sesuai dengan *keyword.* Oleh karena itu, diperlukannya pendekatan yang menjadi alternatif pencarian dari *keyword* yang pengguna tuliskan meskipun tidak sama persis dengan nama yang ada di buku.

Dalam mengatasi tantangan tersebut, terdapat sebuah pendekatan yang sering digunakan adalah menggunakan Algoritma *Levenshtein Distance*. Algoritma ini memainkan peran penting dalam menilai tingkat kesamaan antara dua String, terutama ketika terdapat kesalahan penulisan atau variasi dalam kata kunci. *Levenshtein Distance* mengukur jumlah operasi yang diperlukan untuk mengubah satu String menjadi String lain (Prakash & Liju, 2020). Sehingga, dapat melakukan analisa secara otomatis untuk mendapatkan rasio yang tinggi dan perhitungan jarak *string* pada data yang serupa untuk membantu menangani kesalahan penulisan pada kata kunci atau perbedaan kecil dalam pengejaan. Misalnya, jika pengguna salah mengetik kata kunci atau ada variasi dalam istilah pencarian, algoritma ini dapat memberikan bobot kesamaan yang memungkinkan mesin pencari untuk menyesuaikan hasil pencarian dan tetap memberikan hasil yang relevan (Daniati dkk., 2022). Penerapan *Levenshtein Distance* dalam penelitian ini ditujukan untuk menemukan solusi yang inovatif dan efisien serta dapat memberikan fleksibilitas tambahan untuk menangani data yang tidak terstruktur, seperti pada kasus buku *Letter-C* yang mungkin mengandung kesalahan penulisan nama pemilik, guna membantu proses pencarian data yang akurat dengan proses yang optimal. Efektivitas ini

diukur melalui dua dimensi: keakuratan pencarian (precision) dan kecepatan respons sistem *(response time).* Menurut Young dkk., (2021) keakuratan pencarian mengacu pada kemampuan sistem untuk menghasilkan hasil yang relevan dan akurat terhadap Query yang diberikan, sementara kecepatan respon menunjukkan seberapa cepat sistem dapat menyediakan hasil tersebut kepada pengguna. Kombinasi dari kedua faktor ini mengukur efektivitas sistem dalam memberikan hasil yang tidak hanya akurat dan relevan, tetapi juga tepat waktu, mencerminkan proses pencarian yang optimal.

Berdasarkan identifikasi masalah yang terjadi di Desa Pandanarum serta kebutuhan solusi yang inovatif dan efisien dalam pengelolaan data *Letter-C,* maka penulis mengangkat judul "Optimalisasi Proses Pencarian Data *Letter-C* Melalui Penerapan Algoritma *Levenshtein Distance* di Desa Pandanarum". Judul ini mencerminkan tujuan penelitian untuk mengembangkan sebuah metode pencarian data yang tidak hanya tepat dan cepat, tetapi juga mudah diadaptasi oleh desa-desa lain. Melalui penerapan Algoritma *Levenshtein Distance,* diharapkan dapat dihasilkan sebuah sistem pencarian data yang dapat mengurangi kesalahan dan meningkatkan efisiensi dalam pengelolaan data administratif desa.

METODE PENELITIAN

Penelitian ini dilaksanakan mulai bulan Oktober 2023 hingga Juli 2024. Rentang waktu ini dilaksanakan untuk memastikan tahap penelitian, termasuk pengumpulan data, analisis, dan penyusunan laporan, dapat diselesaikan secara menyeluruh. Lokasi penelitian yang dipilih oleh penulis adalah Desa Pandanarum, yang berada di bagian selatan Kabupaten Blitar, Jawa Timur. Jenis penelitian yang dipilih untuk dilaksanakan adalah penelitian deskriptif. Menurut Ramadhan penelitian deskriptif dirancang untuk menggambarkan karakteristik subjek penelitian atau fenomena yang sedang diobservasi tanpa bermaksud untuk menetapkan hubungan sebab-akibat antara variabel.

HASIL DAN PEMBAHASAN

1. Perhitungan Algoritma Levinstainss Distance

Setelah proses *Pre-Processing,* langkah berikutnya adalah menginisialisasikan string 1 dan string 2 untuk digunakan untuk menghitung jarak antara string pencarian dan string dalam database.

Tabel 1 Data inisialisasi string 1 dan string 2 setelah *pre-processing*

NOMOR	***String* 1**	***String* 2**
2859	["simari", "hidayati", "simarihidayati"]	["agus", "siswanto", "agussiswanto"]
3014	["simari", "hidayati", "simarihidayati"]	["asnari", "mutowiyah", "asnarimutowiyah"]
3807	["simari", "hidayati", "simarihidayati"]	["imam", "sumali", "imamsumali"]
3665	["simari", "hidayati", "simarihidayati"]	["indarwati", "sumaji", "indarwatisumaji"]
3745	["simari", "hidayati", "simarihidayati"]	["juprianto", "sumartin", "jupriantosumartin"]
3013	["simari", "hidayati", "simarihidayati"]	["moh", "latif", "dista", "vidia", "anggraini", "mohlatifdistavidiaanggraini"]

NOMOR	*String* 1	*String* 2
3008	["simari", "hidayati", "simarihidayati"]	["rinasih", "hadi", "prayitno", "rinasihhadiprayitno"]
3015	["simari", "hidayati", "simarihidayati"]	["slamet", "karmilah", "slametkarmilah"]
3010	["simari", "hidayati", "simarihidayati"]	["sopyan", "napiatun", "sopyannapiatun"]
3893	["simari", "hidayati", "simarihidayati"]	["sukari", "surati", "sukarisurati"]
3795	["simari", "hidayati", "simarihidayati"]	["sumadi", "wiji", "utami", "sumadiwijiutami"]
3873	["simari", "hidayati", "simarihidayati"]	["sumaji", "sulastri", "sumajisulastri"]
3009	["simari", "hidayati", "simarihidayati"]	["sumari", "nurhidayati", "sumarinurhidayati"]
3016	["simari", "hidayati", "simarihidayati"]	["sumariana", "iksan", "romadhon", "sumarianaiksanromadhon"]
3695	["simari", "hidayati", "simarihidayati"]	["sumarji", "winarsih", "sumarjiwinarsih"]
3631	["simari", "hidayati", "simarihidayati"]	["supardi"]
2839	["simari", "hidayati", "simarihidayati"]	["suraji", "sumarti", "surajisumarti"]
3011	["simari", "hidayati", "simarihidayati"]	["sutrisno", "desvita", "rosida", "areni", "sutrisnodesvitarosidaareni"]
3012	["simari", "hidayati", "simarihidayati"]	["suyut", "sutini", "suyutsutini"]
3017	["simari", "hidayati", "simarihidayati"]	["yasmadi", "ayem", "lestari", "yasmadiayemlestari"]

Data tabel yang telah disiapkan akan dihitung satu per satu yaitu mengukur jumlah minimum operasi (penambahan, penghapusan, atau penggantian karakter) yang diperlukan untuk mengubah satu string menjadi string lainnya (Prakash & Liju, 2020). Sebelum membangun matriks jarak lengkap, pertama-tama menghitung jarak Levenshtein untuk masing-masing pasangan token berdasarkan karakter. Ini akan memberikan pemahaman tentang seberapa mirip setiap pasangan token.

Tabel 2 Matrix perhitungan pada tiap token

	3009		
	sumari	Nurhidayati	Sumarinurhidayati
simari	1	8	12
hidayati	6	3	9
simarihidayati	9	5	4

Perhitungan dilakukan dengan membandingkan setiap karakter dalam string pencarian dengan karakter dalam string di database, dan hasilnya digunakan untuk menentukan kesamaan antara kedua string tersebut.

Tabel 3 Perhitungan jarak pada setiap label

NOMOR	*String* 1	*String* 2	Jarak
2859	["simari", "hidayati", "simarihidayati"]	["agus", "siswanto", "agussiswanto"]	[[6,5,9],[7,6,10],[13,10,11]]
3014	["simari", "hidayati", "simarihidayati"]	["asnari", "mutowiyah", "asnarimutowiyah"]	[[3,8,11],[6,8,12],[11,10,11]]
3807	["simari", "hidayati", "simarihidayati"]	["imam", "sumali", "imamsumali"]	[[3,2,6],[6,6,7],[11,10,9]]
3665	["simari", "hidayati", "simarihidayati"]	["indarwati", "sumaji", "indarwatisumaji"]	[[6,2,11],[4,6,10],[9,10,10]]
3745	["simari", "hidayati", "simarihidayati"]	["juprianto", "sumartin", "jupriantosumartin"]	[[8,3,12],[7,6,13],[10,9,13]]
3013	["simari", "hidayati", "simarihidayati"]	["moh", "latif", "dista", "vidia", "anggraini", "mohlatifdistavidiaanggraini"]	[[5,5,5,5,7,22], [8,6,6,5,7,21], [12,12,12,11,11,20]]
3008	["simari", "hidayati", "simarihidayati"]	["rinasih", "hadi", "prayitno", "rinasihhadiprayitno"]	[[4,4,7,14], [6,5,6,14], [10,11,11,11]]
3015	["simari", "hidayati", "simarihidayati"]	["slamet", "karmilah", "slametkarmilah"]	[[5,7,9], [6,8,12], [11,10,12]]
3010	["simari", "hidayati", "simarihidayati"]	["sopyan", "napiatun", "sopyannapiatun"]	[[5,7,11],[6,7,11],[11,11,12]]
3893	["simari", "hidayati", "simarihidayati"]	["sukari", "surati", "sukarisurati"]	[[2,3,8],[6,5,8],[10,9,7]]
3795	["simari", "hidayati",	["sumadi", "wiji", "utami", "sumadiwijiutami"]	[[2,4,4,11], [6,6,6,12], [9,12,12,8]]

NOMOR	*String* 1	*String* 2	Jarak
	"simarihidayati"]		
3873	["simari", "hidayati", "simarihidayati"]	["sumaji", "sulastri", "sumajisulastri"]	[[2,4,9],[6,6,10],[10,11,8]]
3009	["simari", "hidayati", "simarihidayati"]	["sumari", "nurhidayati", "sumarinurhidayati"]	[[1,8,12],[6,3,9],[9,5,4]]
3016	["simari", "hidayati", "simarihidayati"]	["sumariana", "iksan", "romadhon", "sumarianaiksanromadhon"]	[[4,5,6,17], [7,6,7,19], [7,12,11,14]]
3695	["simari", "hidayati", "simarihidayati"]	["sumarji", "winarsih", "sumarjiwinarsih"]	[[2,4,10],[6,6,12],[9,10,8]]
3631	["simari", "hidayati", "simarihidayati"]	["supardi"]	[[3],[6],[9]]
2839	["simari", "hidayati", "simarihidayati"]	["suraji", "sumarti", "surajisumarti"]	[[3,2,7],[6,5,9],[10,8,8]]
3011	["simari", "hidayati", "simarihidayati"]	["sutrisno", "desvita", "rosida", "areni", "sutrisnodesvitarosidaareni"]	[[6,6,5,5,21], [8,7,7,6,21], [11,12,10,11,18]]
3012	["simari", "hidayati", "simarihidayati"]	["suyut", "sutini", "suyutsutini"]	[[5,4,9],[6,7,9],[11,11,12]]
3017	["simari", "hidayati", "simarihidayati"]	["yasmadi", "ayem", "lestari", "yasmadiayemlestari"]	[[4,6,4,12], [6,6,6,13], [11,12,12,13]]

Setelah semua jarak ditemukan maka langkah selanjutnya yaitu mencari nilai minimum pada masing-masing hasil *Levinstains Distance* kemudian dipisahkan nilai untuk mempermudah dalam perhitungan total jarak minimumnya dari setiap token dan total minimum inilah yang disebut *Levinstains Distance.*

Tabel 4 Hasil penjumlahan nilai minimum

NOMOR	Jarak	Nilai Minimum	*Levinstains Distance*
2859	[[6,5,9],[7,6,10],[13,10,11]]	5,6,10	21
3014	[[3,8,11],[6,8,12],[11,10,11]]	3,6,10	19
3807	[[3,2,6],[6,6,7],[11,10,9]]	2,6,9	17
3665	[[6,2,11],[4,6,10],[9,10,10]]	2,4,9	15
3745	[[8,3,12],[7,6,13],[10,9,13]]	3,6,9	18
3013	[[5,5,5,5,7,22], [8,6,6,5,7,21], [12,12,12,11,11,20]]	5,5,11	21
3008	[[4,4,7,14], [6,5,6,14], [10,11,11,11]]	4,5,10	19
3015	[[5,7,9], [6,8,12], [11,10,12]]	5,6,10	21
3010	[[5,7,11],[6,7,11],[11,11,12]]	5,6,11	22
3893	[[2,3,8],[6,5,8],[10,9,7]]	2,5,7	14
3795	[[2,4,4,11], [6,6,6,12], [9,12,12,8]]	2,6,8	16
3873	[[2,4,9],[6,6,10],[10,11,8]]	2,6,8	16
3009	[[1,8,12],[6,3,9],[9,5,4]]	1,3,4	8
3016	[[4,5,6,17], [7,6,7,19], [7,12,11,14]]	4,6,7	17
3695	[[2,4,10],[6,6,12],[9,10,8]]	2,6,8	16
3631	[[3],[6],[9]]	3,6,9	18
2839	[[3,2,7],[6,5,9],[10,8,8]]	2,5,8	15
3011	[[6,6,5,5,21], [8,7,7,6,21], [11,12,10,11,18]]	5,7,10	22
3012	[[5,4,9],[6,7,9],[11,11,12]]	4,6,11	21
3017	[[4,6,4,12], [6,6,6,13], [11,12,12,13]]	4,6,11	21

Pada tahap ini string 1 dan string 2 sudah mendapatkan nilai *Levinstains Distance* yang Dimana nilai ini adalah hasil akhir yang sebenarnya sudah bisa dipakai untuk mendapatkan hasil yang sesuai berdasarkan nilai terendah, nilai bisa terlihat ketika diurutkan dari yang terkecil hingga terbesar.

Tabel 5 Mengurutkan jarak terkecil

NOMOR	*String* 2	*Levinstains Distance*
3009	["sumari", "nurhidayati", "sumarinurhidayati"]	8

NOMOR	*String* 2	*Levinstains Distance*
3893	["sukari", "surati", "sukarisurati"]	14
3665	["indarwati", "sumaji", "indarwatisumaji"]	15
2839	["suraji", "sumarti", "surajisumarti"]	15
3795	["sumadi", "wiji", "utami", "sumadiwijiutami"]	16
3873	["sumaji", "sulastri", "sumajisulastri"]	16
3695	["sumarji", "winarsih", "sumarjiwinarsih"]	16
3807	["imam", "sumali", "imamsumali"]	17
3016	["sumariana", "iksan", "romadhon", "sumarianaiksanromadhon"]	17
3745	["juprianto", "sumartin", "jupriantosumartin"]	18
3631	["supardi"]	18
3014	["asnari", "mutowiyah", "asnarimutowiyah"]	19
3008	["rinasih", "hadi", "prayitno", "rinasihhadiprayitno"]	19
2859	["agus", "siswanto", "agussiswanto"]	21
3013	["moh", "latif", "dista", "vidia", "anggraini", "mohlatifdistavidiaanggraini"]	21
3015	["slamet", "karmilah", "slametkarmilah"]	21
3012	["suyut", "sutini", "suyutsutini"]	21
3017	["yasmadi", "ayem", "lestari", "yasmadiayemlestari"]	21
3010	["sopyan", "napiatun", "sopyannapiatun"]	22
3011	["sutrisno", "desvita", "rosida", "areni", "sutrisnodesvitarosidaareni"]	22

Pada tahap ini, dari perhitungan sudah mendapatkan hasil nilai *Levenshtein Distance* untuk setiap pasangan *string.* Selanjutnya, akan menggunakan persentase kemiripan untuk menguji hasil Pengujian Hasil.

2. Pengujian Hasil

Setelah proses perhitungan Algoritma *Levenshtein Distance*, langkah berikutnya adalah menguji hasil perhitungan tersebut untuk memastikan akurasi dan keandalannya. Pengujian ini dilakukan dengan menggunakan berbagai data input untuk melihat bagaimana algoritma berfungsi dalam berbagai scenario (Priyatna, 2023). Untuk memastikan akurasi dan keandalan algoritma, pengujian dilakukan dengan berbagai data uji lainnya untuk melihat bagaimana algoritma berfungsi dalam berbagai skenario pencarian.

Langkah-langkah pengujian hasil:

1. Menentukan data uji

Pemilihan data uji harus mencakup berbagai variasi teks untuk memastikan algoritma dapat menangani berbagai skenario. Misalnya, string input "Simari Hidayati" dibandingkan dengan beberapa nama pemilik dalam database untuk melihat tingkat kemiripan yang dihasilkan.

Data uji harus mencakup berbagai panjang dan kompleksitas *string* untuk memastikan algoritma dapat menangani perbedaan-perbedaan ini dengan efektif. Dengan menggunakan data uji yang beragam, kemampuan algoritma dalam mengenali kemiripan dalam situasi yang berbeda dapat dievaluasi, seperti nama yang sangat mirip, nama yang berbeda dengan beberapa karakter yang sama, dan nama yang benar-benar berbeda.

2. Menghitung persentase kemiripan

Setelah data uji ditentukan, langkah berikutnya adalah menghitung persentase kemiripan antara *string* pencarian dan *string* dalam *database*. Persentase kemiripan dihitung menggunakan rumus yang memperhitungkan jarak *Levenshtein* antara dua *string* dan panjang maksimum dari kedua *string* tersebut. Langkah ini memberikan ukuran kuantitatif dari seberapa mirip dua string tersebut, yang memudahkan analisis lebih lanjut. Pada tahap ini sudah mendapatkan hasil tetapi karena perlunya mengoptimalisasikan data pencarian dan ini masih memiliki beberapa permasalahan yang dimana ketika nilai *Levinstainss Distance* memiliki nilai yang sama maka masih menjadi sebuah kebingungan karena bisa saja kata kunci yang tepat berada di urutan bawah sebab masih bersifat acak, maka dari itu dibutuhkan *similarity percent* untuk mendapatkan hasil yang lebih detail dengan mengubah untuk dicari persentase tertinggi.

$$Similarity\ Percent\ = \left(1 - \frac{Jarak\ Levenshtein}{max(panjang\ string\ 1, panjang\ string\ 2)}\right) \times 100$$
$$Similarity\ Percent\ = \left(1 - \frac{8}{34}\right) \times 100$$
$$Similarity\ Percent\ = (1 - 0{,}2353) \times 100$$
$$Similarity\ Percent\ = 0{,}7647 \times 100$$
$$Similarity\ Percent\ = 76{,}47\ \%$$

Hasil perhitungan *persentase* kemiripan memberikan pandangan yang lebih jelas tentang seberapa dekat dua *string* tersebut. Misalnya, string "Simari Hidayati" dibandingkan dengan "SUMARI NURHIDAYATI" menghasilkan persentase kemiripan yang cukup tinggi, menunjukkan bahwa kedua string tersebut memiliki kesamaan yang signifikan. Sebaliknya, jika dibandingkan dengan "AGUS SISWANTO", persentase kemiripan akan rendah, menunjukkan perbedaan yang jelas antara kedua string.

3. Menentukan keputusan

Berdasarkan persentase kemiripan yang telah dihitung, keputusan diambil apakah dua string tersebut "Mirip" atau "Tidak Mirip". Keputusan ini didasarkan pada indeks yang telah ditentukan sebelumnya: 0% - 39%: Tidak mirip, 40% - 59%: Sedikit mirip, 60% - 79%: Cukup mirip, dan 80% - 100%: Sangat mirip. Langkah ini penting untuk mengklasifikasikan hasil berdasarkan tingkat kemiripan yang diukur.

Indeks ini digunakan untuk memberikan interpretasi yang lebih jelas tentang hasil perhitungan. Nilai 0% - 39% dianggap "Tidak mirip" karena jarak *Levenshtein* menunjukkan perbedaan yang signifikan antara *string-string* tersebut. Nilai 40% - 59% dianggap "Sedikit mirip" karena ada beberapa kesamaan, namun perbedaan masih cukup besar. Nilai 60% - 79% dianggap "Cukup mirip" karena tingkat kesamaan yang lebih

tinggi dan perbedaan yang lebih sedikit. Nilai 80% - 100% dianggap "Sangat mirip" karena *string-string* tersebut hampir identik, dengan sangat sedikit perbedaan.

Tabel 6 Menentukan keputusan penguji

NOMOR	*String* 2	*Levinstains Distance*	Persentase Kemiripan	Keputusan
3009	["sumari", "nurhidayati", "sumarinurhidayati"]	8	76.47%	Cukup Mirip
3893	["sukari", "surati", "sukarisurati"]	14	50.00%	Sedikit Mirip
3665	["indarwati", "sumaji", "indarwatisumaji"]	15	48.28%	Sedikit Mirip
2839	["suraji", "sumarti", "surajisumarti"]	15	48.28%	Sedikit Mirip
3695	["sumarji", "winarsih", "sumarjiwinarsih"]	16	46.67%	Sedikit Mirip

Pengujian hasil ini memberikan pandangan yang lebih jelas tentang tingkat kemiripan antara string pencarian dan *string* dalam *database*. Persentase kemiripan yang lebih tinggi menunjukkan bahwa *string-string* tersebut lebih mirip, sedangkan persentase yang lebih rendah menunjukkan bahwa *string-string* tersebut kurang mirip. Keputusan akhir tentang apakah dua *string* tersebut "Mirip" atau "Tidak Mirip" didasarkan pada persentase kemiripan yang telah dihitung.

Pengujian hasil menggunakan berbagai data input membantu memastikan akurasi dan keandalan algoritma, sehingga dapat diandalkan dalam berbagai skenario pencarian dan pencocokan teks. Dengan langkah-langkah ini, hasil perhitungan Algoritma *Levenshtein Distance* telah diuji menggunakan persentase kemiripan. Hasil ini menunjukkan tingkat kesamaan antara string pencarian dan string dalam database, membantu dalam pengambilan keputusan akhir apakah dua *string* tersebut "Mirip" atau "Tidak Mirip". Sesuai yang diharapkan dapat memberikan hasil yang lebih cepat dan akurat, meminimalkan kesalahan, dan meningkatkan kepuasan pengguna. Perhitungan Algoritma *Levenshtein Distance* yang dilakukan secara sistematis memastikan bahwa hasil pencarian mencerminkan kemiripan teks secara akurat.

3. Implementasi API *JSON*

Setelah proses perhitungan Algoritma *Levenshtein Distance* dan pengujian hasil, langkah berikutnya adalah mengimplementasikan API *JSON* untuk memungkinkan akses dan penggunaan algoritma ini melalui antarmuka web. Implementasi API *JSON* ini penting agar sistem dapat berinteraksi dengan aplikasi lain dan menyediakan hasil pencarian yang efisien.

Gambar 1. URL Penggunaan Enpoint API *JSON*

Pertama, harus membuat *endpoint API* di server yang akan menerima permintaan *(request)* dan mengembalikan hasil dalam format *JSON*. *Endpoint* ini diatur menggunakan *framework* web seperti Django untuk membuat API *JSON* sesuai yang diharapkan.

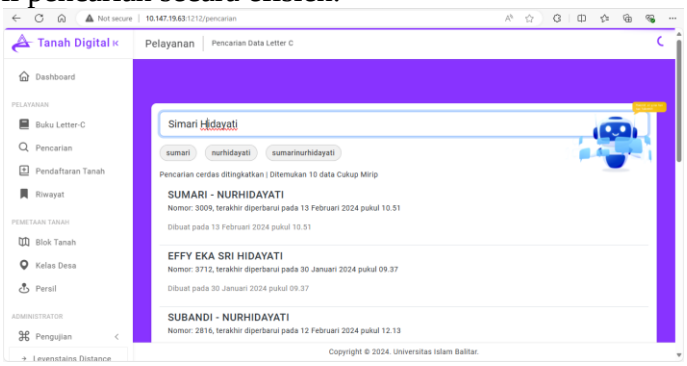
Tabel 7 Penjelasan *Endpoint JSON*

Field	Tipe Data	Deskripsi
count	Integer	Jumlah total hasil yang ditemukan.
enhancement	Boolean	Menunjukkan apakah pencarian cerdas digunakan.
token_terdekat	List of *String*s	Daftar token yang paling mendekati input pencarian. Digunakan untuk memperbaiki salah ketik
modus	*String*	Kategori hasil pencarian berdasarkan tingkat kemiripan.
results	List of Objects	Daftar hasil pencarian. Setiap objek dalam daftar ini mewakili satu entri hasil.
label	*String*	Label unik untuk setiap hasil.
teks_input	*String*	Teks input yang digunakan untuk pencarian.
teks_database	*String*	Teks dari database yang dibandingkan dengan input.
jarak	Integer	Jarak Levenshtein antara teks input dan teks database.
persen	Float	Persentase kemiripan antara teks input dan teks database.
keputusan	*String*	Keputusan akhir berdasarkan persentase kemiripan (Misal: "Cukup Mirip", "Tidak Mirip",dll).
token_input	List of *String*s	Token-token yang dihasilkan dari teks input.
token_database	List of *String*s	Token-token yang dihasilkan dari teks database.
token_terdekat	List of *String*s	Token-token yang paling mendekati hasil pencarian.
data	Object	Informasi tambahan terkait hasil pencarian.
data.id	Integer	ID unik untuk setiap hasil dalam database.

Field	Tipe Data	Deskripsi
data.nomor	*String*	Nomor unik untuk setiap hasil dalam database.
data.nama_pemilik_saat_ini	*String*	Nama pemilik saat ini dari data dalam database.
data.create_at	*String* (Datetime)	Tanggal dan waktu ketika data dibuat.
data.update_at	*String* (Datetime)	Tanggal dan waktu ketika data diperbarui.

Endpoint API dibuat di server untuk menerima permintaan *(request)* dan mengembalikan hasil dalam format JSON. Penggunaan *framework* web seperti Django memungkinkan pembuatan API *JSON* yang dapat diakses oleh aplikasi lain.

Untuk menerapkan API *JSON* ke web, sebuah server web diatur untuk menangani permintaan dari klien web. Penggunaan *framework* seperti Django dapat mempermudah dalam membangun *endpoint* yang menangani permintaan *GET* atau *POST* dari klien. Hasil yang dikembalikan dalam format *JSON* memungkinkan klien web untuk menampilkan hasil pencarian secara efisien.



Gambar 2 Implementasi API *JSON* ke web

Implementasi API *JSON* tidak terbatas pada web saja, namun juga dapat diintegrasikan ke dalam aplikasi mobile seperti Android dan iOS. Aplikasi mobile dapat mengirim permintaan ke *endpoint* API dan menampilkan hasil yang dikembalikan dalam format *JSON*. *Framework* seperti *Retrofit* untuk Android dan *Alamofire* untuk iOS dapat digunakan untuk mengintegrasikan API *JSON* ke dalam aplikasi mobile.

Gambar 3 Integrasi API *JSON* ke dalam aplikasi mobile.

Dengan langkah-langkah ini, implementasi API *JSON* diharapkan dapat memberikan hasil pencarian yang cepat dan akurat, memudahkan integrasi dengan aplikasi lain, serta meningkatkan kepuasan pengguna. Implementasi yang baik juga memastikan bahwa API dapat diakses dengan mudah oleh berbagai platform, baik web maupun mobile, sehingga memberikan fleksibilitas yang tinggi dalam penggunaannya.

4. Pembahasan

Algoritma *Levenshtein Distance* efektif dalam meningkatkan proses pencarian data kepemilikan tanah di Desa Pandanarum. Algoritma ini mampu mengatasi tantangan yang dihadapi dalam pencarian data manual seperti kesalahan pengetikan dan variasi ejaan. Penerapan Algoritma *Levenshtein Distance* terbukti meningkatkan kecepatan pencarian data, mengurangi waktu yang diperlukan dibandingkan dengan metode manual (Priyatna, 2023). Algoritma ini juga mampu menangani kesalahan pengetikan dan variasi ejaan, sehingga meningkatkan akurasi pencarian informasi. Dengan demikian, penelitian ini berhasil mencapai tujuannya untuk meningkatkan efisiensi dan akurasi pencarian data (Rofiqih dkk., 2022).

Namun, penelitian ini juga memiliki beberapa kelemahan. Implementasi algoritma memerlukan sumber daya manusia yang terampil dalam teknologi informasi. Kompleksitas data yang tidak terstruktur seperti buku *Letter-C* tetap menjadi tantangan meskipun algoritma dapat mengatasi sebagian besar kesalahan pengetikan dan variasi ejaan (Setiawan dkk., 2022). Selain itu, penelitian ini terbatas pada satu desa, sehingga hasilnya mungkin tidak langsung berlaku untuk tempat lain dengan kondisi yang berbeda. Algoritma ini juga memiliki kelemahan dalam menangani nama yang sangat panjang karena perhitungannya berdasarkan panjang karakter, yang dapat mempengaruhi kinerja dan akurasi pencarian (Prakash & Liju, 2020). Namun, algoritma ini sangat cocok untuk digunakan dalam memberikan rekomendasi dengan nilai jarak tertinggi (Oktavia Nahampun, 2023). Meskipun demikian, penelitian ini membuka peluang untuk pengembangan lebih lanjut dalam aplikasi teknologi informasi di bidang administrasi kepemilikan tanah, memberikan solusi bagi permasalahan serupa di desa-desa lain,

sekaligus meningkatkan pemahaman tentang aplikasi teknologi informasi dalam konteks lokal (Equi dkk., 2023).

KESIMPULAN

Implementasi Algoritma *Levenshtein Distance* dalam pencarian data *Letter-C* di Desa Pandanarum terbukti efektif meningkatkan efisiensi dan akurasi. Algoritma ini mengukur jarak antara dua string dengan menghitung jumlah minimum operasi yang diperlukan untuk mengubah satu string menjadi string lainnya, seperti penambahan, penghapusan, dan penggantian karakter. Dalam konteks ini, algoritma membantu mengidentifikasi entri database yang paling mirip dengan teks pencarian pengguna, meskipun terdapat variasi ejaan atau kesalahan pengetikan, sehingga sangat bermanfaat dalam menangani data yang tidak terstruktur dan memperbaiki kelemahan metode manual yang memakan waktu dan rentan terhadap kesalahan.

Hasil penelitian menunjukkan bahwa penerapan Algoritma *Levenshtein Distance* meningkatkan akurasi pencarian data. Algoritma ini berhasil mengidentifikasi dan memperbaiki kesalahan pengetikan dengan lebih baik, sehingga akurasi pencarian meningkat dari pencari satu per satu dari 480 data menjadi bisa dicari dalam satu waktu tanpa membuka buku secara langsung. Kecepatan pencarian juga meningkat secara signifikan, mengurangi waktu yang diperlukan untuk menemukan data yang relevan dari rata-rata 3 hari untuk pencarian menjadi kurang satu jam dalam setiap pencarian. Selain itu, algoritma ini berhasil mengurangi risiko kesalahan dalam pencarian data, dengan tingkat kesalahan yang langsung diperbaiki saat itu juga. Dengan demikian, penerapan algoritma ini tidak hanya membuat proses pencarian lebih efisien dan cepat, tetapi juga meningkatkan keandalan dan ketepatan data yang ditemukan.

Secara keseluruhan, penelitian ini berhasil mencapai tujuan utamanya untuk mengoptimalkan proses pencarian data kepemilikan tanah di Desa Pandanarum, serta memberikan kontribusi nyata dalam pengembangan teknologi informasi di bidang administrasi tanah.

DAFTAR PUSTAKA

Abdul Wahid, R. (2021). Digitalisasi Registrasi Desa (Letter C) Tanah. 6.

Arifin, N. Y., & Prasetyo, E. (2021). Perancangan Sistem Web Semantik Database Dokumen Qa.

Daniati, Y. N., Zukarnain, I. A., & Nurfitri, K. (2022). Pada Sistem Pencarian Data Buku Berbasis Web. Http://Studentjournal.Umpo.Ac.Id/Index.Php/Komputek

Equi, M., Mäkinen, V., Tomescu, A. I., & Grossi, R. (2023). On The Complexity Of String Matching For Graphs. Acm Transactions On Algorithms, 19(3). Https://Doi.Org/10.1145/3588334

Forel, D., Benz, T., & Pennington, W. (2005). Seismic Data Processing With Seismic Unx.

Hum, M., & Lubis, A. A. (2022). Hukum Agrari: Kajian Komprehensif (Vol. 426). Pustaka Prima. Www.Pustaka-Prima.Com

Hunter, J., Dale, D., & Firing, E. (2017). 2020 - Matplotlib.

Kornienko, D. V., Mishina, S. V., Shcherbatykh, S. V., & Melnikov, M. O. (2021). Principles Of Securing Restful Api Web Services Developed With Python Frameworks. Journal Of Physics: Conference Series, 2094(3). Https://Doi.Org/10.1088/1742-6596/2094/3/032016

Oktavia Nahampun, H. (2023). Resolusi : Rekayasa Teknik Informatika Dan Informasi Penerapan Algoritma Levenstein Distance Untuk Pencarian Judul Pada Aplikasi Lagu-Lagu Nasional. Media Online, 3(4), 282–286. Https://Djournals.Com/Resolusi

Petty, T., Hannig, J., Huszar, T. I., & Iyer, H. (2022). A New String Edit Distance And Applications. Algorithms, 15(7). Https://Doi.Org/10.3390/A15070242

Prakash, P., & Liju, M. (2020). Algorithm To Derive Shortest Edit Script Using Levenshtein Distance Algorithm.

Priyatna, R. D. (2023). Implementasi Algoritma Levenstein Distance Dan Algoritma Knutt Morris Prattdalamfitur Word Completionpada Search Engine. Implementasi Algoritma Levenstein Distance Dan Algoritma Knutt Morris Prattdalamfitur Word Completionpada Search Engine, 1, 145–156.

Ramadhan, M. (2021). Metode Penelitian (A. A. Effendy, Ed.). Cipta Media Nusantara. Https://Books.Google.Com/Books?Hl=En&Lr=&Id=Ntw_Eaaaqbaj&Oi=Fnd&Pg=Pr1&Dq=Penelitian+Deskriptif&Ots=F3lk2mqsbx&Sig=Bzyqzkusy-Csdkphflmpgba9vsy

Rofiqih, A., Siambaton, M. Z., & Haramaini, T. (2022). Penerapan Algoritma Levenshtein String Pada E-Arsip Kecamatan Pagar Merbau. Sudo Jurnal Teknik Informatika, 1(1), 1–7. Https://Doi.Org/10.56211/Sudo.V1i1.1

Romzi, M., & Kurniawan, B. (2020). Pembelajaran Pemrograman Python Dengan Pendekatan Logika Algoritma (Nomor 2).

Saabith, A. S., Vinothraj, T., & Fareez, M. (2021). A Review On Python Libraries And Ides For Data Science. Dalam International Journal Of Research In Engineering And Science (Ijres) Issn. Www.Ijres.Org

Santi, I. H., Febrinita, F., & Puspitasari, W. D. (2023). Journal Of Intelligent Decision Support System (Idss) Engineering Design Business Process Modelling Letter C Land Data Archiving System With Software Requirement Specifications Approach. Dalam Journal Of Intelligent Decision Support System (Idss) (Vol. 6, Nomor 4).

Setiawan, E., Santi, H., & Budiman, S. N. (2022). Sistem Pengelolaan Dan Pengamanan Arsip Data Letter C Desa (Studi Kasus : Kantor Desa Gondang). Dalam Jurnal Mahasiswa Teknik Informatika) (Vol. 6, Nomor 2).

Sinaga, A., & Nuraisana. (2021). Implementasi Algoritma Brute Force Dalam Pencarian Menu Pada Aplikasi Pemesanan Coffee (Studi Kasus : Tanamera Coffee). 4(1), 6–15.

Soepandi, H., & Widodo, H. (2021). Perancangan Sistem Informasi Pertanahan Buku C Desa Berbasis Web Di Desa Satriyan Kec.Tersono Kabupaten Batang. Http://Ejournal.Stmik-Wp.Ac.Id

Sugeng, T. A. (2021). Fungsi Buku Tanah Desa Sebagai Landasan Yuridis Awal Alat Bukti Hak Kepemilikan Atas Tanah The Function Of The Village Land Book As The Initial Jurisdictional Evidence Tool Of Land Ownership Rights. Jurnal Cermin, 5(2), 1273–1284. Https://Doi.Org/10.22219/Cermin.V5i2.1273

Trisman M, Haripuddin, & Sanatang. (2023). Pengembangan Sistem Deteksi Kemiripian Ta Berbasis Algoritma Rabin Karp & Levensthein Naskah Skripsi Mahasiswa Jtik. Etnik : Jurnal Ekonomi-Teknik Informasi Artikel, 2(2), 1–131.

Vamsi, K. M., Lokesh, P., Reddy, K. N., & Swetha, P. (2021). Visualization Of Real World Enterprise Data Using Python Django Framework. Iop Conference Series: Materials Science And Engineering, 1042(1), 012019.

Https://Doi.Org/10.1088/1757-899x/1042/1/012019