Nicole M. Ramirez Mulero

801-19-3940

Prof. Orozco

14 de mayo de 2025

I. Establecimiento del Problema

La Distancia de Levenshtein es un algoritmo que mide las similitudes entre dos cadenas. En esencia, es un número que indica la diferencia entre dos cadenas de caracteres [3]. Cuanto mayor sea este número, mayor será la diferencia entre ambas cadenas de caracteres. Este cálculo considera la inserción (lo que se debe añadir para que ambas cadenas sean iguales), la eliminación (lo que se debe eliminar para que ambas cadenas sean iguales) y el reemplazo (lo que se debe cambiar para que ambas cadenas sean iguales) [1]. Este algoritmo tiene aplicaciones en el procesamiento de texto, específicamente en la limpieza de datos, la búsqueda y la autocorrección [1]. Este proyecto se centra en la autocorrección ortográfica y en cómo este algoritmo puede predecir lo que el usuario intenta escribir y sugerir una alternative bien escrita de la palabra que el usuario intenta escribir. Por lo tanto, el problema que este proyecto busca resolver es el "**Método de autocorrección ortográfica utilizando el algoritmo de distancia de Levenshtein**".

II. Antecedentes y Metodologías

La comparación de cadenas de caracteres es una tarea muy importante en diversas disciplinas [1]. En la Bioinformática la comparación se usa para detector anomalías en las cadenas de AND y ARN para la detección de mutaciones e incluso la detección de ciertos tipos

de cáncer. Para la ciencia de datos, es una tarea fundamental en la búsqueda de información y recuperación de datos. También en las redes sociales es una tarea que se usa para la recomendación de contenido, ya que comparando títulos y descripciones se pueden recomendar publicaciones (películas, títulos de artículos, etc.) que tengan similitudes con lo que ya el usuario ha buscado. En el ámbito academia también es usado puesto a que herramientas de detección de plagio se elaboran con algoritmos de comparación de cadenas de caracteres [1]. En conclusión, es una tarea multidisciplinaria necesitada en muchos aspectos.

Este trabajo se enfoca en la corrección de ortografía. En casi todo procesador de texto tiene por necesidad un comparador de cadenas de caracteres para poder realizar su funcionamiento de manera efectiva. Navegadores de internet comparan el input del usuario para hacer búsquedas en sus bases de datos, y programas como Microsoft Word utilizan la comparación para sugerirle correcciones de ortografía al usuario. En este caso se implementó un comportamiento similar utilizando el algoritmo de Levenshtein.

III. El Algoritmo

El algoritmo de Levenshtein, también conocido como distancia de Levenshtein o distancia de edición, fue diseñado por Vladimir Levenshtein, matemático soviético, en 1965 [2]. Es un método para medir la similitud entre dos cadenas. Este algoritmo calcula la diferencia entre dos cadenas calculando cuántas ediciones se necesitan para que sean idénticas. Estas ediciones se basan en añadir, reemplazar o eliminar un carácter. Se suma un punto por cada edición, y cuanto mayor sea la puntuación, más distintas serán las dos cadenas [1].

La función principal de este algoritmo es identificar las diferencias y similitudes entre dos líneas de texto. Para ello, utiliza un enfoque de programación dinámica con una matriz de edición. La

programación dinámica es una técnica en la que un problema se divide en subproblemas, se guardan los resultados y luego se optimizan los subproblemas para encontrar la solución general, que generalmente tiene que ver con encontrar el rango máximo y mínimo de la consulta algorítmica.

Cada celda de la matriz contiene la distancia mínima de edición entre las dos cadenas hasta ese punto [3]. La matriz se puede construir de la siguiente manera: si ambos caracteres son iguales, solo se toma el número de la línea anterior en diagonal. Si son diferentes, se toma el número menor superior, a la izquierda o en diagonal a la izquierda, y si hay caracteres adicionales, se suma la cantidad de caracteres adicionales existentes. A continuación, se muestra un ejemplo.

	.	g	a	t	o
.	0	1	2	3	4
g	1	0	1	2	4
t	2	3	1	3	4
a	3	4	3	2	3
o	4	4	4	3	2

La distancia de Levenshtein entre «gato» y «gtao» es 2.

En este trabajo, se utiliza el algoritmo de distancia de Levenstein, que cuenta con un método de autocorrección ortográfica. La idea es un programa que sugiere una palabra aproximada para corregir errores ortográficos. El algoritmo de Levenstein se ejecuta mediante programación dinámica en una matriz construida con ambas cadenas de texto (esta función se

denomina levenshteinFullMatrix). La idea principal de esta aplicación es que el usuario ingrese una palabra y el programa verifique si está escrita correctamente y ofrezca una sugerencia aproximada de lo que probablemente quiso decir el usuario. Para el alcance de este proyecto, las únicas palabras que el programa puede identificar son un banco de 700 nombres de animales [5]. LevenshteinFullMatrix es utilizado por la función CheckBestMatch, que compara cada palabra del archivo CSV con la ingresada por el usuario, buscando la palabra con la distancia de Levenshtein más baja. Si encuentra la misma palabra, le indica al usuario que la ortografía es correcta; de lo contrario, el programa sugiere una palabra para corregirla.

La implantación del código en Jupiter Notebook puede ser encontrada en el siguiente repositorio de Github: : https://github.com/Nicole-M-Ramirez/LD-Spelling-Checking

IV. Análisis de Complejidad

A. Complejidad temporal del algoritmo de distancia de Levenshtein:

[1] O(1)

[2] O(1)

[3] O(m n)

[4] O (m)

[5] O(1)

[6] O(n)

[7] O(1)

[8] [9] O(m , n)

[10] O(1)

[11] O(1)

[12] O(1)

[13] O(1)

[14] O(1)

La complejidad general de este algoritmo es T(m, n) = O(m * n)

B. Complejidad temporal para CheckBestMatch:

[1] O(1)

[2] O(1)

[3] O(k)

[4] O (m * n)

[5] O (1)

[6] O (1)

[7] O(1)

[8] O(1)

[9] O(1)

[10] O(1)

[11] O(1)

[12] O(1)

[13] O(1)

[14] O(1)

La complejidad general de este algoritmo es T(m, n) = O(l * (m * n))

V. Pruebas de Usabilidad

La manera en que escribe cada usuario y los errores que pueda cometer está sujeto a cada usuario. Es algo subjetivo donde no todas las palabras tendrán el mismo porcentaje de error. Por ende, para comprobar cuan efectivo es la algoritmo, se usaron 699 palabras (nombres de animales) Al cuando cada palabra se le aplico un 10%, 30%, 50%, 75% de error. Para aplicar este porcentaje de error se utilizó la herramienta de inteligencia artificial de Google, Gemini. Con esas palabras generadas, se les aplico la función CheckBestMatch y se contó cuantas veces el algoritmo dio una sugerencia errónea.

En un 10% de error, el algoritmo se equivocó en el 0.29% (2 palabras), lo que deja un 99.71% de exactitud. En un 30% de error, el algoritmo se equivocó en el 01.3% (9 palabras), lo que deja un 98.70% de exactitud. En un 50% de error, el algoritmo se equivocó en el 16.89% (2 palabras), lo que deja un 83.12% de exactitud. En un 75% de error, el algoritmo se equivocó en el 67.38% (471 palabras), lo que deja un 32.62%% de exactitud.

VI. Herramientas computacionales

Este programa se implementó con Python 3.0. Para una mejor organización, se utilizó Jupiter Notebook. Se utilizó la biblioteca csv para poder leer la Animal_names.csv que contiene

todos los nombres de animales que se utilizaron para las pruebas. Tambien se uso la librería numpy para crear la matriz de edicion. Los nombres de animales fueron sacados de [6].

VII. Conclusiones

El Algoritmo de Levenshtein es un algoritmo que permite la rápida comparación de 2 cadenas de texto. Es un algoritmo intuitivo y flexible que puede ser utilizado para muchas ramas, desde la Bioinformática hasta los procesamientos de lenguaje natural. La implementación en este caso fue capaz de generar exitosamente recomendaciones de arreglo de ortografía dependiendo del input del usuario.

VI. Bibliografía

[1] Liju, P. (2022). Algorithm to derive shortest edit script using Levenshtein distance algorithm.

[2] Mehta, A., Salgond, V., Satra, D., & Sharma, N. (2021). Spell correction and suggestion using Levenshtein distance. *Int Res J Eng Technol*, *8*(8), 1977-1981.

[3] Po, D. K. (2020). Similarity based information retrieval using Levenshtein distance algorithm. *Int. J. Adv. Sci. Res. Eng*, *6*(04), 06-10.

[4] Nadhia Nurin Syarafina, Jozua Ferjanus Palandi, & Nira Radita. (2021). Designing a word recommendation application using the Levenshtein Distance algorithm. *Matrix: Jurnal Manajemen Teknologi Dan Informatika*, *11*(2), 63–70. https://doiorg.uprrp.idm.oclc.org/10.31940/matrix.v11i2.2419

[5] GeeksforGeeks, “Introduction to Levenshtein distance,” *GeeksforGeeks*. https://www.geeksforgeeks.org/introduction-to-levenshtein-distance

[6] “A simple csv of animal names - scraped from https://a-z-animals.com/animals/,” *Gist*. https://gist.github.com/EyeOfMidas/311e77b8b8c2f334fc8bdaf652c1f47f