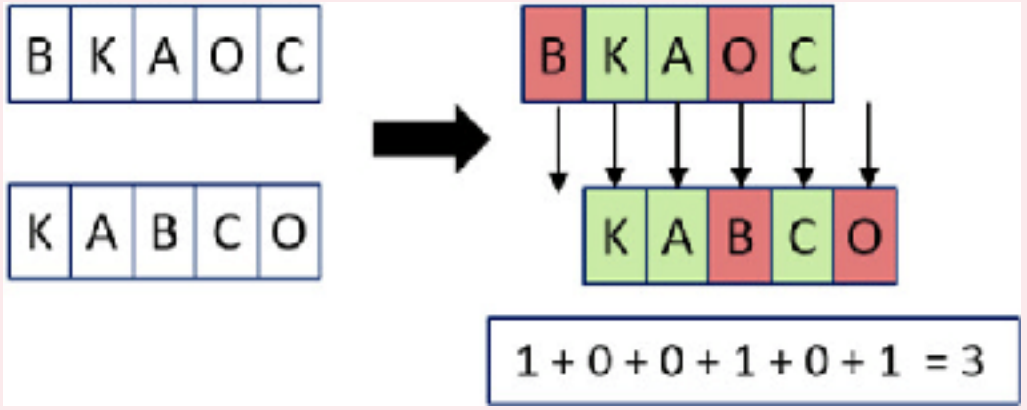
Corrector Ortográfico Utilizando Algoritmo de Levenshtein

Nicole M. Ramirez Mulero
Prof. Orozco
CCOM 6050
Mayo 14 de 2025

Introducción

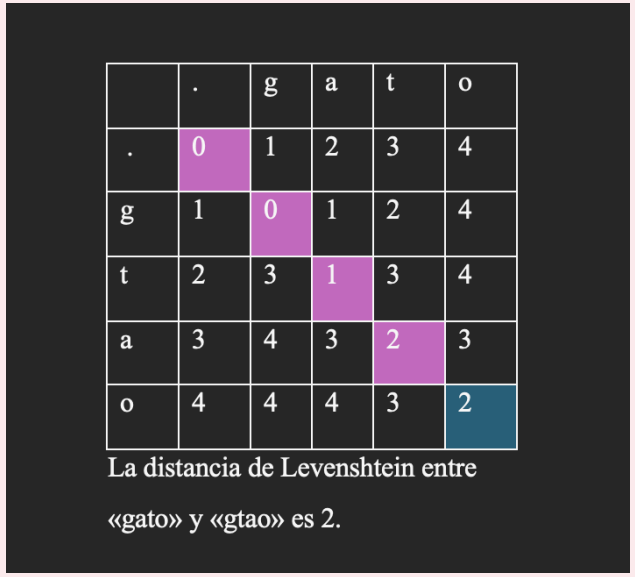
Problema Por Solucionar:
Método de autocorrección ortográfica utilizando el algoritmo de distancia de Levenshtein



- La tare ade comparer cadenas de caracteres es una utilizada atravez de muchas aplicacion por muchas diciplinas
- Se basa en tomar 2 cadenas de strings y comparer sus similitudes y diferencias
- La comparación de strings es una operación central en varios entornos, vital para manejar variaciones y errores en el texto.
- El emparejamiento aproximado de cadenas es un método aplicado en búsqueda de texto, reconocimiento de patrones y procesamiento de señales.
- Para esta tarea se an creado multiples herraminetoas para realizarla: Bag distance, Smith-Waterman distance, Longest common sub-string (LCS), etc.

El Algoritmo de Levenstein

- El algoritmo fue formulado y explicado por el matemático soviético Vladimir Levenshtein en 1965.
- El algoritmo de Levenshtein es una parte del método de Aproximación de Emparejamiento de Cadenas. Es un caso especial de la Distancia de Edición. donde el costo de las operaciones (inserción, eliminación, sustitución) es de 1 unidad.
- Calcula la distancia o similitud entre dos strings (una string fuente 's' y un string objetivo 't') basándose en el número mínimo de operaciones de edición.



	.	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.

Initialization

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

$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(source[i]) \\ D[i,j-1] + \text{ins-cost}(target[j]) \\ D[i-1,j-1] + \text{sub-cost}(source[i], target[j]) \end{cases}$$

Recurrence Relation:

```
For each  i = 1…M
      For each  j = 1…N
                          { D(i-1,j) + 1
          D(i,j)= min     { D(i,j-1) + 1
                          { D(i-1,j-1) +   2; if X(i) ≠ Y(j)
                                           0; if X(i) = Y(j)
```

Termination:

```
D(N,M) is distance
```

Código

```python
import numpy as np

def levenshteinFullMatrix(str1, str2):
    #Obtener tamaño de strings
    len_str1 = len(str1)
    len_str2 = len(str2)

    #Crear matriz inicializada con 0's
    leven_matrix = np.zeros((len_str1 + 1, len_str2 + 1),dtype="int")

    #inicializar primera linea i columna con numeros de 0 a len_str1 y 0 a len_str2 respectivamente
    leven_matrix[0, :] = np.arange(len_str2 + 1)
    leven_matrix[:, 0] = np.arange(len_str1 + 1)


    #Llenar matriz
    for i in range(1, len_str1 + 1):
        for j in range(1, len_str2 + 1):
            #si los caracteres son iguales, se copia numero diagonal
            if str1[i - 1] == str2[j - 1]:
                leven_matrix[i][j] = leven_matrix[i - 1][j - 1]

            #si los caracteres no son iguales se elije el valor de costo minimo entre insersion, borado o sustitucion
            else:
                inser = leven_matrix[i][j - 1]
                delete = leven_matrix[i - 1][j]
                sustitution = leven_matrix[i - 1][j - 1]

                leven_matrix[i][j] = 1 + min(inser, delete, sustitution)

    #devuelve la distancia de edicion enrte los 2 strings
    return leven_matrix[len_str1][len_str2]
```

Código

```
def CheckBestMatch(data, word):
    bestMatchLD = levenshteinFullMatrix(data[0], word)
    bestMatchWord = data[0]

    for i in range(1,len(data)):
        LD = levenshteinFullMatrix(data[i], word)

        if (LD <= bestMatchLD):
            bestMatchLD = LD
            bestMatchWord = data[i]

    if(bestMatchLD == 0 and bestMatchWord != ""):
        print("Your spelling is correct")
    elif(bestMatchWord != ""):
       print("Word fix suggestion:",bestMatchWord)
    else:
       print("No matches found")
```

```
import csv
def getDataSet(filename):
    csvFile = []
    with open(filename, mode ='r')as file:
      item = csv.reader(file)
      for lines in item:
            csvFile.append(lines[1])

    csvFile.pop(0)
    return csvFile
```

Pruebas

- Se le pidió a Gemini (inteligencia artificial de Google) dañar 699 palabras en un 10%, 30%, 50% y 75% de la palabra.
- Estas palabras dañadas se procesaron utilizando el algoritmo para medir cuantas veces el algoritmo sugería una palabra errónea dependiendo del nivel de errores ortográficos.
- A las palabras se les cambio, añadió o borro letras para medir el error ortográfico.

Porcentaje de error ortografico en la palabra	Porcentaje de equivocaciones del algortimo	Porcentaje de Exactitud
10%	0.29% (2 palabras)	99.71%
30%	1.3% (9 palabras)	98.70%
50%	16.89% (118 palabras)	83.12%
75%	67.38% (471 palabras)	32.62%

Análisis de Complejidad

```python
import numpy as np

def levenshteinFullMatrix(str1, str2):
    #Obtener tamaño de strings
    len_str1 = len(str1)
    len_str2 = len(str2)

    #Crear matriz inicializada con 0's
    leven_matrix = np.zeros((len_str1 + 1, len_str2 + 1),dtype="int")

    #inicializar primera linea i columna con numeros de 0 a len_str1 y 0 a len_str2 respectivamente
    leven_matrix[0, :] = np.arange(len_str2 + 1)
    leven_matrix[:, 0] = np.arange(len_str1 + 1)


    #Llenar matriz
    for i in range(1, len_str1 + 1):
        for j in range(1, len_str2 + 1):
            #si los caracteres son iguales, se copia numero diagonal
            if str1[i - 1] == str2[j - 1]:
                leven_matrix[i][j] = leven_matrix[i - 1][j - 1]

            #si los caracteres no son iguales se elije el valor de costo minimo entre insersion, borado o sustitucion
            else:
                inser = leven_matrix[i][j - 1]
                delete = leven_matrix[i - 1][j]
                sustitution = leven_matrix[i - 1][j - 1]

                leven_matrix[i][j] = 1 + min(inser, delete, sustitution)

    #devuelve la distancia de edicion enrte los 2 strings
    return leven_matrix[len_str1][len_str2]
```

O(1)

O(n)

O(m)

O(1)

T(n, m) = O(n * m)

Análisis de Complejidad

```
def CheckBestMatch(data, word):
    bestMatchLD = levenshteinFullMatrix(data[0], word)
    bestMatchWord = data[0]

    for i in range(1,len(data)):
        LD = levenshteinFullMatrix(data[i], word)

        if (LD <= bestMatchLD):
            bestMatchLD = LD
            bestMatchWord = data[i]

    if(bestMatchLD == 0 and bestMatchWord != ""):
        print("Your spelling is correct")
    elif(bestMatchWord != ""):
       print("Word fix suggestion:",bestMatchWord)
    else:
       print("No matches found")
```

O(1)

O(l)

O(n * m)

O(1)

O(l * n * m)

Conclusión

- El algoritmo de Levensthein es un algoritmo que permite la rapida comparacion de 2 cadenas de caracteres.
- Es un algoritmo intuitivo y flexible que puede ser utilizado para muchas ramas, desde la Bioinformática hasta los procesamientos de lenguaje natural.
- Fue capaz de brindar una recomendación acertada en la gran mayoría de los casos con un 10% y un 30% de error, y en una buena porción de los casos en el 50% de error

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