PRACTICA 5 – *LEVENSHTEIN* – RAÚL FERNÁNDEZ ESPAÑA UO278036

En esta practica trabajaremos la programación dinámica mediante el cálculo de la distancia de levenshtein.

Se llama distancia del levenshtein al número mínimo de operaciones que hay que hacer para que dos palabras o cadenas sean iguales.
Siguiendo los parámetros solo esta permitido insertar un carácter borrar o cambiar un carácter por otro.

SOLUCIÓN RECURSIVA
En 1965, Levenshtein formuló (d es distanciaLevenshtein):

d(i,j)= d(i-1,j-1) si cad1[i] == cad2[j]
d(i,j)= 1 + min (d[i-1,j-1],d[i,j-1],d[i-1,j]) si cad1[i] != cad2[j]

En la practica dada se da como ejemplo las palabras cad1=”BARCAZAS” y cad2=”ABRACADABRA”:
Con el siguiente resultado

TRAZA CON cad1=”BARCAZAS” y cad2=”ABRACADABRA”:

			B	A	R	C	A	Z	A	S
		j=*0*	j=*1*	j=*2*	j=*3*	j=*4*	j=*5*	j=*6*	j=*7*	j=*8*
	i=*0*	0	1	2	3	4	5	6	7	8
A	i=*1*	1	1	1	2	3	4	5	6	7
B	i=*2*	2	1	2	2	3	4	5	6	7
R	i=*3*	3	2	2	2	3	4	5	6	7
A	i=*4*	4	3	2	3	3	3	4	5	6
C	i=*5*	5	4	3	3	3	4	4	5	6
A	i=*6*	6	5	4	4	4	3	4	4	5
D	i=*7*	7	6	5	5	5	4	4	5	5
A	i=*8*	8	7	6	6	6	5	5	4	5
B	i=*9*	9	8	7	7	7	6	6	5	5
R	i=*10*	10	9	8	7	8	7	7	6	6
A	i=*11*	11	10	9	8	8	8	8	7	**7 (SOL)**

Al probar nuestro algoritmo con las mismas palabras para comprobar el resultado hemos obtenido una matriz exactamente igual.

Como

ABRACADABRA	BARCAZAS							
0	1	2	3	4	5	6	7	8
1	1	1	2	3	4	5	6	7
2	1	2	2	3	4	5	6	7
3	2	2	2	3	4	5	6	7
4	3	2	3	3	3	4	5	6
5	4	3	3	3	4	4	5	6
6	5	4	4	4	3	4	4	5
7	6	5	5	5	4	4	5	5
8	7	6	6	6	5	5	4	5
9	8	7	7	7	6	6	5	5
10	9	8	7	8	7	7	6	6
11	10	9	8	8	8	8	7	7

podemos observar la matriz imprimida por nuestro código se corresponde con el resultado.

La complejidad temporal del algoritmo:

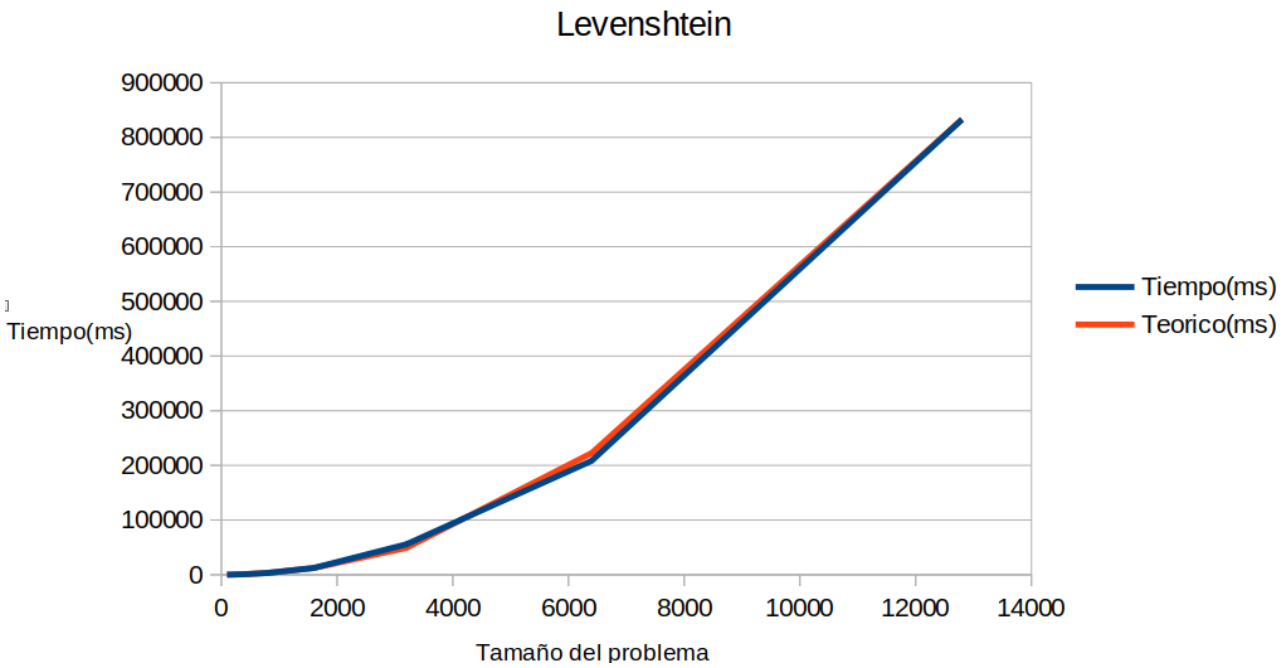
Primero tomaremos valores experimentales del algoritmo, para analizarlos visualmente, posteriormente compararemos si los resultados esperados se corresponden con, los resultados esperados teóricamente y su complejidad.

Tras la toma de datos del algoritmo con diferentes palabras e incrementando el tamaño del problema el resultado de la toma de datos es el siguiente.

N(tam. Problema)	Tiempo(ms)	nVeces
100	74	1000
200	181	1000
400	862	1000
800	3051	1000
1600	12307	1000
3200	55631	1000
6400	208295	1000
12800	832758	1000

(No he pegado directamente por que al usar libreoffice no queda igual)

Este es el resultado de la toma de datos para el tamaño del problema que se puede observar, del cual obtenemos las consecuentes mediciones de tiempos para un tamaño del problema 1000.



Como podemos observar gráficamente en la toma de datos los resultados de la toma de datos experimental se acercan mucho a los resultados esperados dentro del marco teórico.

Gráficamente según la toma de datos recopilada parece que el algoritmo tiene una complejidad temporal $O(n^2)$.

Teóricamente el algoritmo consta de una complejidad $O(n^2)$, el algoritmo posee cuatro bucles, dos de ellos no anidados con una complejidad O(n). Sin embargo existen dos bucles los cuales se encuentran anidados, ejecutando operaciones en su interior O(1), al repetirse n veces estos bucles su complejidad es O(n), debido a que se encuentran anidados la complejidad resultante es de $O(n^2)$.

Por tanto el algoritmo desempeña correctamente su función ya que como podemos observar las mediciones y la complejidad son los esperados, coincidiendo los resultados experimentales con los teóricos con un margen de error mínimo.

Para realizar pruebas y asegurarnos de la implementación del algoritmo se ha realizado el método generadorPalabras, el cual nos genera palabras de manera aleatoria para el calculo de la distancia de Levenshtein.