MEMORIA
PROYECTO ALT

Autores: *Sergi Albiach Caro, Manel Angresola Navarro,*

Stéphane Díaz-Alejo León & Antonio Martínez Leal (4CO11)

ÍNDICE

1.-INTRODUCCIÓN

En el siguiente documento se describen las actividades realizadas a lo largo de un proyecto que se puede dividir en dos partes: un motor de recuperación de información, el cual se llevó a cabo en la asignatura de *Sistemas de almacenamiento y recuperación de información*; y un sistema de búsqueda aproximada, que hemos completado en el curso actual de *Algorítmica*. Nuestro foco de atención se colocará en la segunda tarea puesto que la primera ya fue explicada en un reporte anterior.

El motor de recuperación contó por nuestra parte con la implementación de las funcionalidades de stemming, multifield, permuterm, utilización de paréntesis y clasificación por ranking.

En cuanto a la segunda tarea, principal objetivo a tratar, hemos realizado las siguientes actividades:

- Implementado las distancias de Levenshtein, Damerau-Levenshtein restringida y Damerau-Levenshtein intermedia.
    - De manera opcional, también hemos implementado una versión general de Damerau-Levenshtein.
- Implementado una versión mejorada de las distancias anteriores teniendo en cuenta un umbral “threshold”.
- Realizado mejoras para evitar cálculos innecesarios cuando la distancia es mayor al umbral establecido.
- Implementado el cálculo de la distancia entre una cadena y un trie.
- Realizado un estudio para determinar qué versiones de las distancias vistas son más rápidas.
- Añadido el código realizado al proyecto de SAR para en caso de no obtener resultados con la palabra de la query, buscar con aquellas que estén a una distancia de edición más cercana hasta obtener resultados.

La primera de las seis tareas se dividió entre los diferentes miembros del equipo, realizando cada distancia por parejas. Hecho esto, el grupo ya estaba familiarizado con la tarea y pudimos pasar a trabajar individualmente, realizando cada persona una de las distancias a completar (las tres versiones con threshold y la versión general). Seguidamente, pusimos todas las implementaciones en común y decidimos trabajar conjuntamente para la elaboración de la distancia cadena-trie (puesto que contaba con una dificultad mayor) y el resto de comprobaciones. De esta forma, todo el proyecto restante fue realizado en grupo menos la penúltima tarea, en la cual volvimos a trabajar en parejas para agilizar el proceso. En el informe se detalla en mayor medida las actividades y personas involucradas.

Por último pero no menos importante, cabe destacar que hemos utilizado GitHub como herramienta de coordinación, almacenando nuestro código en un repositorio privado que nos permitía colaborar todos a la vez.

2.-INFORME

2.1.-TAREA 1

En la tarea número 1 implementamos mediante programación dinámica las distancias: Levenshtein, Damerau-Levenshtein restringida y Damerau-Levenshtein intermedia. Nuestro equipo decidió que, al ser sencilla su implementación, podíamos realizar su versión con reducción espacial, sustituyendo la matriz empleada en los métodos por las columnas necesarias.

De tal forma, tras haber entendido la distancia levenshtein entre todos, Damerau-Levenshtein restringida fue realizada por Manel y Stéphane, mientras que Damerau-Levenshtein intermedia se llevó a cabo por Sergi y Antonio.

2.1.1.- TAREA OPCIONAL

Una vez comprendidos los modelos básicos era más factible realizar la versión general. Se partió del modelo que calcula la distancia Damerau-Levenshtein intermedia con una variable de valor constante de 1 (cte = 1) con la idea de utilizarlo para valores mayores.

Para conseguir nuestro objetivo se dividió el valor cte en dos variables de forma que (|u| + |v|) <= cte. Este cambio permitió unificar diversas instrucciones if y se precisó de utilizar una matriz entera en vez de una fila al contar con un mayor campo de búsqueda.

Esta parte fue realizada por Antonio.

2.2.-TAREA 2

En la tarea número 2 se nos requería implementar una versión mejorada de las distancias anteriormente mencionadas mediante la adición de un parámetro, *threshold*, que parase el cómputo de aquellas distancias superior a este. Nuestro equipo decidió implementar esta versión mejorada sobre las que poseían reducción espacial, por lo que también cuentan con este aspecto.

Las implementaciones con threshold se realizaron de la siguiente manera: Levenshtein por Stéphane, Damerau-Levenshtein restringida por Manel y Damerau-Levenshtein intermedia por Sergi.

2.3.-TAREA 3

Para realizar la tarea 3 hemos creado la clase SpellSuggester que genera una vocabulario con las palabras del Quijote (quijote.txt) y un método *suggest()* que recibe el término a comparar, el tipo de distancia a aplicar y el threshold a tener en cuenta y devuelve un diccionario donde para cada distancia de edición tiene asociada la lista de palabras que se encuentran a esa distancia.
Para evitar comparar cadenas con una distancia de edición mayor que una cota optimista, se restan las longitudes de la palabra a comparar con cada una de las palabras del vocabulario y si esta es mayor que la cota, no se realiza el cómputo de dicha distancia.

El método suggest fue implementado por todo el equipo.

2.4.-TAREA 4

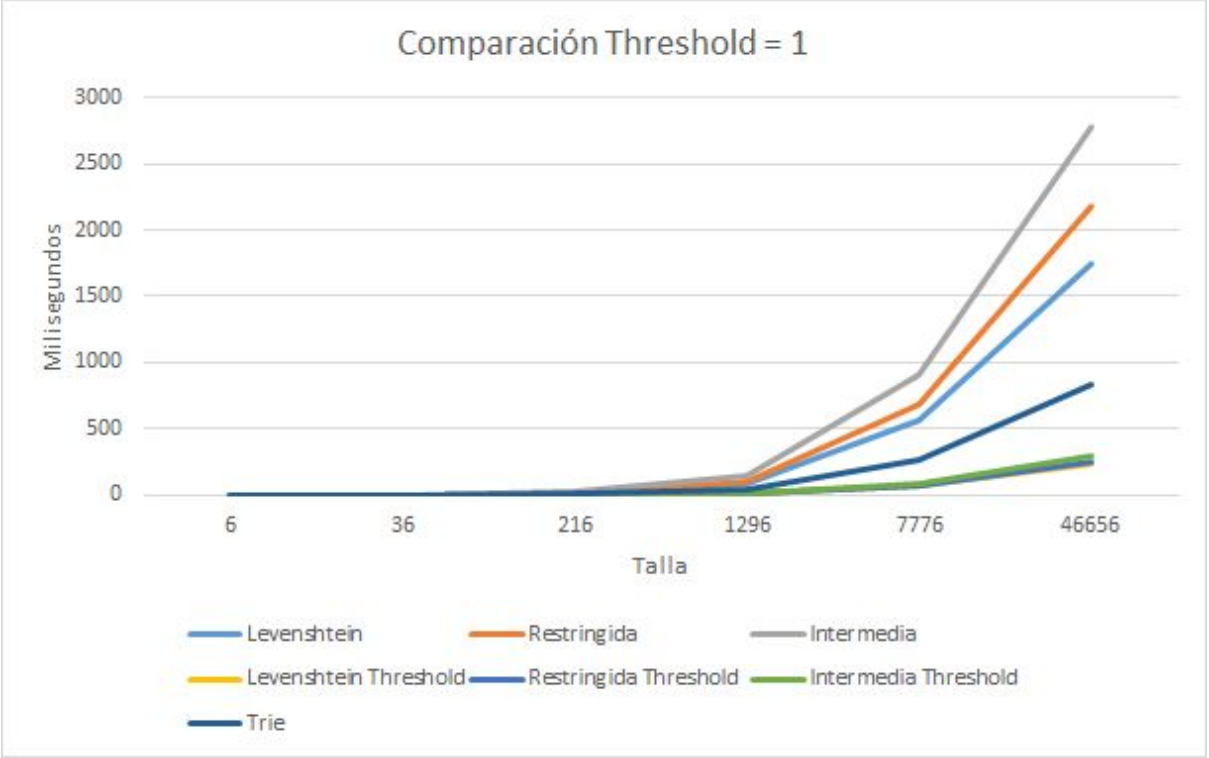
La tarea número 4 consistía en implementar una versión que calculase la distancia cadena-trie. Para esto, y teniendo en cuenta que no nos pareció trivial, decidimos implementarlo sin tener en cuenta la reducción espacial utilizando una matriz, donde las filas son cada una de las letras de la cadena y las columnas son cada uno de los estados del trie. Al acabar de computar toda la matriz, recorremos los hijos del nodo padre y cuando encontramos un nodo final, añadimos el coste que tiene la matriz al diccionario resultante y apilamos sus hijos para seguir comprobando todas las palabras del trie.

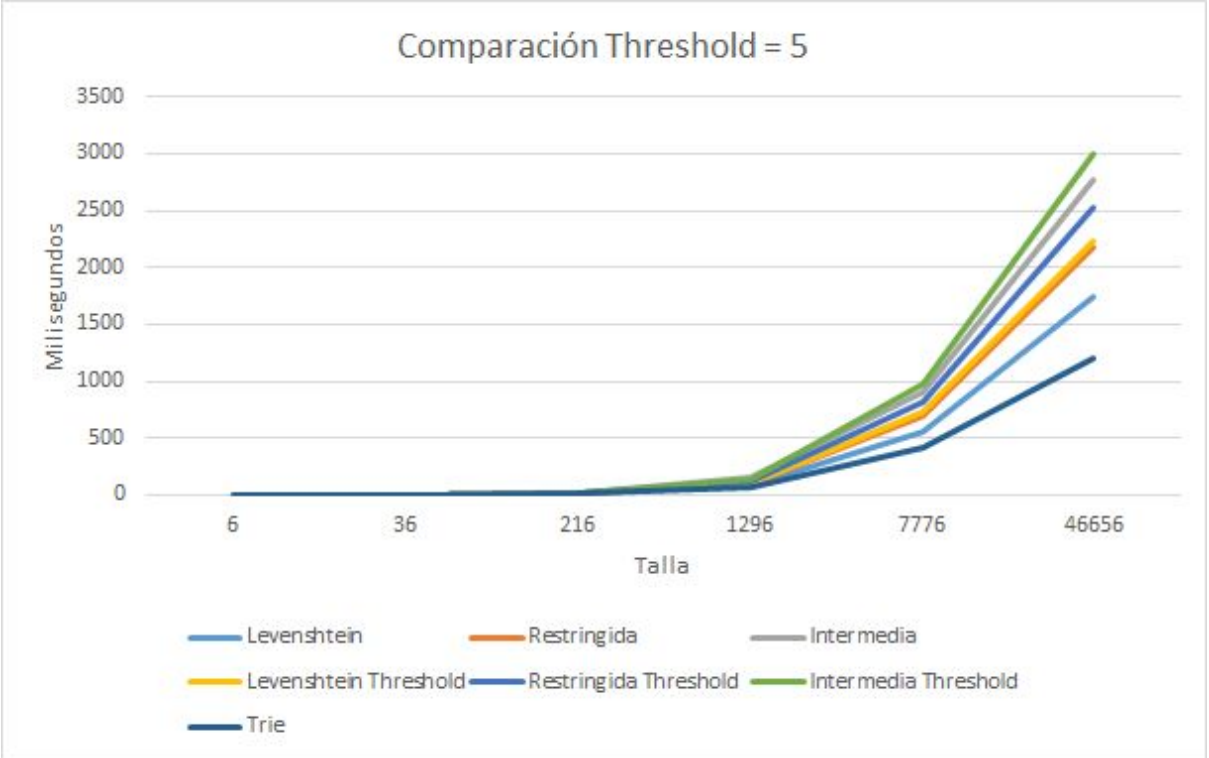
La distancia cadena-trie fue implementada por todo el equipo.

2.5.-TAREA 5

En la tarea número 5 se nos requería realizar un estudio experimental sobre los tiempos que tardaban las distintas implementaciones de distancias que fuimos programando a lo largo del proyecto de esta asignatura. Para ello, nuestro grupo ha generado un conjunto de palabras con errores ortográficos al azar (mediante código), el cuál será nuestro conjunto de prueba. Cada palabra de este conjunto será contrastada frente a diccionarios de distintas tallas utilizando los distintos métodos. En el caso de las versiones de los métodos que poseen un parámetro *threshold*, serán ejecutados diversas veces con valores distintos. Finalmente, de los tiempos medidos, se calcularán la media, mediana y desviación típica.

Con todo lo explicado con anterioridad, los resultados obtenidos han sido los siguientes:





Podemos observar en los resultados, como se esperaba, que los métodos que utilizan la técnica del *threshold* proporcionan tiempos más cortos que sus versiones sin ellas cuando les permite parar el cómputo (*threshold* = 1). Sin embargo, cuando se ha de explorar todo el espacio de búsqueda (*threshold* = 5), los tiempos son mayores que sus versiones sin esta técnica, esto pensamos que es causado por mantenerse dentro de la diagonal principal.

A su vez, se puede advertir que la versión de distancia Levenshtein base se ejecuta más rápido que las versiones de Damerau restringida e intermedia, tanto con threshold como sin él. Nuestro razonamiento es que tarda menos debido a que tiene en cuenta menos casos. Sin embargo, cabe remarcar, que las tres versiones pueden proporcionar para la misma palabra distancias distintas. Esto se debe a los casos que contemplan cada una, por lo que el conjunto de palabras que se recomiendan con cada uno de estos métodos será el mismo, pero ordenado de manera distinta, algo a tener en cuenta.

Respecto al método con *trie*, podemos dilucidar que cuanto se ha de explorar todo el espacio de búsqueda, es sin duda, el método más rápido, pero cuando la técnica que usa el *threshold* puede proporcionar una ventaja al acortar cálculos (*threshold* = 1), los métodos sin *trie* obtienen mejores resultados, siendo mejor el método base de Levenshtein, por lo anteriormente citado.

Para finalizar, podemos concluir que el método que se ejecuta en menos tiempo cuando se explora todo el espacio de búsqueda es el que contrasta **cadena-trie**, pero cuando se puede emplear la técnica del *threshold*, el mejor es **Levensthein**.

Esta parte del código fue realizada en dos bloques, la adición automática de errores y la generación del nuevo vocabulario fue realizada por Manel y Sergi. Por otra parte, el código asociado a el cálculo de los valores estadísticos fue realizado por Antonio y Stéphane.

2.6.-TAREA 6

Para la última tarea, se nos pide incorporar el trabajo que hemos realizado en las tareas anteriores en el proyecto de SAR del curso pasado, implementando las clases de SpellSuggest y TrieSpellSuggest.

De esta forma, hemos empezado modificando el indexador “SAR_Indexer.py” para que al ejecutarse con el argumento “-G” o “--suggest”, guarde un objeto de la clase TrieSpellSuggest construido con el vocabulario indexado, para poder así acceder directamente a este objeto cada vez que vayamos a realizar una query posteriormente.

A la hora de realizar la query, hemos modificado “SAR_SpellSuggest.py” para añadir el argumento “-B” o “--busq”, que indica al programa que cada vez que no encuentre una palabra en el diccionario indexado, busque una lista de palabras similares a una misma distancia. Hemos elegido como métodos de búsqueda disponibles Levenshtein con threshold (“levenshtein_thres”) y Levenshtein-trie (“levenshtein-trie”) puesto que son los motivos expuestos en el apartado anterior. Esto se implementa en “SAR_lib.py”, con los métodos “get_posting” y “get_posting_similares”, donde en el caso anterior, buscaría palabras con una distancia desde 1 hasta un máximo de 5, para devolver un conjunto razonable de palabras similares.

A continuación, se indica un ejemplo de ejecución:

```
python .\SAR_Indexer.py -G .\corpora\2015 index.txt
python .\SAR_Searcher.py -B levenshtein_thres index.txt
```

Esta tarea fue realizada por todo el equipo simultáneamente.

APÉNDICE

Tabla 1.- Métodos sin threshold

Método	Talla	Media	Mediana	Desv. típica
Levenshtein	6	0.000207	0.000202	5.185577e-05
Levenshtein	36	0.001572	0.001551	0.000471
Levenshtein	216	0.011254	0.011545	0.002600
Levenshtein	1296	0.082292	0.082031	0.021949
Levenshtein	7776	0.564062	0.53125	0.182946
Levenshtein	46656	1.74375	1.75	0.503949
Restringida	6	0.000276	0.000267	7.552540e-05
Restringida	36	0.002089	0.001906	0.000719
Restringida	216	0.014718	0.014648	0.004385
Restringida	1296	0.104687	0.09375	0.033366
Restringida	7776	0.6875	0.6875	0.202764
Restringida	46656	2.176562	2.109375	0.655544
Intermedia	6	0.000364	0.000357	0.000105
Intermedia	36	0.002765	0.002556	0.001049
Intermedia	216	0.020052	0.018229	0.008450
Intermedia	1296	0.147656	0.148437	0.047193
Intermedia	7776	0.904687	0.921875	0.282069
Intermedia	46656	2.775	2.765625	0.832483

Tabla 2.- Método Levenshtein con threshold

Talla	Threshold	Media	Mediana	Desv. típica
6	1	1.314761e-05	2.264638e-06	3.285923e-05
6	2	2.745287e-05	1.932981e-06	6.463073e-05
6	3	4.667321e-05	2.593819e-06	8.978032e-05

6	4	7.300777e-05	2.163011e-06	0.000114
6	5	0.000144	9.042091e-05	0.000145
36	1	0.000125	3.372290e-05	0.000234
36	2	0.000263	0.000149	0.000387
36	3	0.000482	0.000219	0.000644
36	4	0.000766	0.000497	0.000688
36	5	0.001320	0.001367	0.000919
216	1	0.001113	0.000633	0.001275
216	2	0.002941	0.002228	0.002709
216	3	0.005463	0.006634	0.004065
216	4	0.009201	0.011545	0.005838
216	5	0.012952	0.013281	0.006680
1296	1	0.010060	0.011545	0.007178
1296	2	0.026374	0.029297	0.017125
1296	3	0.048921	0.052083	0.025976
1296	4	0.081510	0.078125	0.033855
1296	5	0.108594	0.109375	0.028049
7776	1	0.075156	0.074219	0.042669
7776	2	0.189844	0.1875	0.094531
7776	3	0.357812	0.359375	0.132297
7776	4	0.553125	0.5625	0.161051
7776	5	0.723437	0.796875	0.160573
46656	1	0.232292	0.210937	0.101854
46656	2	0.615625	0.578125	0.256859
46656	3	1.090625	1.085937	0.338583
46656	4	1.667187	1.75	0.430698
46656	5	2.232812	2.4375	0.539705

Tabla 3.- Método Damerau-Levenshtein restringido con threshold

Talla	Threshold	Media	Mediana	Desv. típica
6	1	1.465448e-05	2.557405e-06	3.678571e-05
6	2	3.329507e-05	2.523373e-06	7.922250e-05
6	3	5.083804e-05	2.241178e-06	9.830257e-05
6	4	8.505627e-05	2.810930e-06	0.000133
6	5	0.000164	9.055512e-05	0.000169
36	1	0.000142	3.371878e-05	0.000273
36	2	0.000295	0.000164	0.000440
36	3	0.000522	0.000247	0.000694
36	4	0.000896	0.000537	0.000805
36	5	0.001521	0.001597	0.001060
216	1	0.001189	0.000672	0.001356
216	2	0.003343	0.002418	0.003149
216	3	0.006089	0.007291	0.004551
216	4	0.010591	0.012912	0.006986
216	5	0.014542	0.015005	0.007649
1296	1	0.011343	0.013672	0.008408
1296	2	0.028536	0.033854	0.017805
1296	3	0.053125	0.055989	0.027858
1296	4	0.09375	0.089844	0.039751
1296	5	0.117187	0.125	0.028384
7776	1	0.078776	0.078125	0.042438
7776	2	0.215625	0.210937	0.098027
7776	3	0.395312	0.398437	0.149878
7776	4	0.645312	0.640625	0.217637
7776	5	0.81875	0.84375	0.188694
46656	1	0.253906	0.21875	0.111434

46656	2	0.660937	0.617187	0.263043
46656	3	1.229687	1.226562	0.394236
46656	4	1.878125	1.992187	0.488030
46656	5	2.51875	2.75	0.606073

Tabla 4.- Método Damerau-Levenshtein intermedio con threshold

Talla	Threshold	Media	Mediana	Desv. típica
6	1	1.630634e-05	2.418748e-06	4.167050e-05
6	2	3.637042e-05	2.526158e-06	8.595026e-05
6	3	6.250909e-05	2.114864e-06	0.000122
6	4	9.542421e-05	2.996643e-06	0.000149
6	5	0.000177	0.000102	0.000181
36	1	0.000161	4.182185e-05	0.000310
36	2	0.000341	0.000199	0.000505
36	3	0.000602	0.000282	0.000802
36	4	0.001049	0.000632	0.000966
36	5	0.001767	0.001909	0.001209
216	1	0.001351	0.000765	0.001537
216	2	0.003784	0.002791	0.003535
216	3	0.007382	0.008764	0.005489
216	4	0.011779	0.013889	0.007651
216	5	0.016811	0.015950	0.008853
1296	1	0.012571	0.015625	0.008729
1296	2	0.033965	0.036458	0.022712
1296	3	0.066901	0.070312	0.033441
1296	4	0.110937	0.105469	0.048134
1296	5	0.136719	0.140625	0.031103
7776	1	0.090755	0.085937	0.053831

7776	2	0.2375	0.234375	0.114564
7776	3	0.475	0.484375	0.177218
7776	4	0.74375	0.75	0.217653
7776	5	0.973437	1.007812	0.230070
46656	1	0.290625	0.265625	0.124844
46656	2	0.785937	0.734375	0.311487
46656	3	1.435937	1.445312	0.448111
46656	4	2.24375	2.265625	0.589425
46656	5	3.003125	3.265625	0.702180

Tabla 5.- Método Levenshtein con threshold y Trie

Talla	**Threshold**	**Media**	**Mediana**	**Desv. típica**
6	1	0.000123	0.000122	2.078154e-05
6	2	0.000159	0.000156	1.743290e-05
6	3	0.000194	0.000190	2.637693e-05
6	4	0.000216	0.000216	3.540135e-05
6	5	0.000233	0.000245	4.037652e-05
36	1	0.000713	0.000647	0.000138
36	2	0.000891	0.000905	0.000129
36	3	0.001093	0.001122	0.000222
36	4	0.001253	0.001310	0.000247
36	5	0.001378	0.001490	0.000297
216	1	0.006324	0.005916	0.001473
216	2	0.007249	0.006836	0.001437
216	3	0.008934	0.008764	0.001845
216	4	0.009490	0.009943	0.001856
216	5	0.010063	0.010440	0.002280
1296	1	0.045312	0.041667	0.009688

1296	2	0.055989	0.058594	0.010433
1296	3	0.064844	0.0625	0.016774
1296	4	0.06875	0.070312	0.016313
1296	5	0.069141	0.074219	0.016942
7776	1	0.273437	0.265625	0.061813
7776	2	0.33125	0.34375	0.068394
7776	3	0.360937	0.375	0.085710
7776	4	0.403125	0.398437	0.112847
7776	5	0.409375	0.390625	0.119977
46656	1	0.83125	0.828125	0.180980
46656	2	1.042187	1.039062	0.260900
46656	3	1.1125	1.125	0.278966
46656	4	1.2375	1.234375	0.338150
46656	5	1.195312	1.1875	0.345962