Memoria prácticas - *ALT*

David Arnal García

Luis Alberto Álvarez Zavaleta

Lugman Sami Ahmad Mansilla

Labores realizadas:

En cuanto a las labores realizadas, se ha decidido realizar todas las tareas colaborativamente todos conectados a la vez usando *LiveShare* de *Visual Studio Code*, por lo que estas han sido hechas por todos los miembros del equipo de manera conjunta.

Descripción resumida del trabajo realizado, indicando si se han implementado todos los requerimientos obligatorios:

Para la realización de este proyecto, hemos realizado las partes obligatorias de cada apartado, esto es:

Tarea 1

Hemos hecho la implementación de las tres distancias:

- Distancia de *Levenshtein*
- Distancia de *Damerau-Levenshtein* restringida
- Distancia de *Damerau-Levenshtein* “intermedia” para *cte* = 1

Basándonos en los algoritmos que hemos encontrado en los apuntes de teoría para la distancia de *Levenshtein*. Para la implementación de la distancia restringida e intermedia hemos tomado de base el primer algoritmo de Levenshtein y lo hemos modificado para obtener las versiones de *Damerau-Levenshtein* ‘restringida’ e ‘intermedia’.

Tarea 2

Mejoramos las distancias de la tarea 1 añadiendo *thresholds* como una variable que fije un umbral en el cual nuestro algoritmo debe dejar de calcular distancias, si este es alcanzado. Esto lo hemos implementado en *test_tarea2.py*.

Tarea 3

En el fichero spellsuggest.py hemos implementado en la función main, la estructura de los resultados de los distintos algoritmos de distancia, utilizando el código que se había proporcionado en *plantilla_generar_resultados.py*.

Usamos el comparador de archivos de *Visual Studio* para comprobar que los resultados proporcionados con los ficheros de resultados que se nos habían proporcionado eran los mismos. Además, también implementamos una cota optimista para evitar la ejecución de las distancias de edición de la tarea 2, si esta cota es mayor que el *threshold*.

Tarea 3: cota optimista basada en contar el número de veces que aparece cada letra en cada palabra (opcional)

En el método *suggest_opt()*, dentro de *spellsuggest.py*, hemos implementado esta nueva cota para la distancia de *Levenshtein*, de forma que como se indica en esta en caso de ser alcanzada no se realice la invocación al método de *Levenshtein*.

Tarea 4

Hemos realizado la comparación entre un par de cadenas usando *tries*, en la cual este podrá contener una cadena más corta siendo esta el prefijo de otra más larga, de esta forma, aprovechando la compactación que ofrece tener un árbol para representar las palabras que tendría un diccionario del que deseamos realizar la

comparación con nuestra cadena. En esta parte, recibiendo una cadena y el *trie*, hemos realizado la comparación hacia atrás de *Levenshtein.*

Tarea 4: Versiones hacia atrás para *Damerau-Levenshtein* restringida e intermedia usando *tries* (opcional)

En *test_tarea4.py* hemos implementado tanto *dp_restricted_damerau_trie* como *dp_intermediate_damerau_trie.*

Tarea 5

Hemos realizado un estudio experimental para determinar qué versiones de las anteriores son las más eficientes. Para este estudio, hemos comprobado para unos datos concretos y los distintos algoritmos implementados, la eficiencia de estos con distintas tallas, distintos valores de *threshold*, analizando distintas palabras, todo ello con los mismos datos para poder comparar entre ellos su eficiencia. Comprobaremos la media, mediana y la desviación típica.

En vista de los resultados obtenidos, expuestos en el anexo, hemos decidido usar *Levenshtein* como distancia predeterminada en *SAR_lib*, debido a que es el que mejores resultados de tiempos nos ha proporcionado para todos los *threshold* que hemos usado para su estudio. Adicionalmente, hemos elegido usar *threshold* '3' como opción predeterminada, ya que da un número de errores aceptables para conseguir sugerencias.

Seleccionado unos valores medios que podemos tomar como referencia para apreciar esta elección, podemos apreciar para *threshold* 3 y una talla de 20.000 que podemos considerar como una talla aceptable los siguientes valores:

	levenshtein	levenshtein-intermedia	levenshtein-restricted	levenshtein-trie	levenshtein-intermedia-trie	levenshtein-restricted-trie
Media	1,208	1,494	1,567	2,739	2,925	3,378
Mediana	1,219	1,531	1,594	2,820	2,984	3,477
Desviación típica	0,3412	0,4022	0,4259	0,3877	0,4371	0,5045

Podemos apreciar cómo se obtienen los mejores tiempos con un ajuste más aproximado a los distintos tamaños de las palabras en base a los resultados con *Levenshtein*.

Tarea 5: generar errores ortográficos (opcional)

En el archivo *medirTiempos.py* hemos generado un método llamado *get_random_terms* que toma un número de palabras aleatorias del vocabulario y aplica transformaciones de borrado, intercambio e inserción de palabras para simular errores ortográficos antes de medir tiempos.

Como podemos apreciar, al aumentar la talla y el *threshold*, la implementación de las distancias de *Levenshtein* con *tries* proporciona mejores tiempos.

Tarea 6

Hemos creado un nuevo archivo llamado *SAR_ALT_spellsuggest.py*, siendo una modificación de *spellsuggest.py*, en el que, en lugar de tomar un archivo, toma directamente el vocabulario. Con esto, podremos guardar en *self.spellsuggester* el vocabulario de cada campo, el cual se guardará en el *bin* generado por el *Indexer* y podrá ser llamado para ser utilizado desde el *Searcher.*

Para ejecutarlo, podremos usar los métodos usados en *SAR*:

py .\SAR_Indexer.py -S -P -M -O -U corpora/2015 2015_index_full.bin

donde *U* es la extensión para incluir el *suggester* en el indexador.

py .\SAR_Searcher.py -C -Q 'word' -E -D 'levenshtein' -H 'threshold' 2015_index_full.bin

donde *-E* activa la búsqueda aproximada, *-D* la distancia y *-H* el *threshold*, por defecto utiliza *Levenshtein* con *threshold* = 3.

Una pequeña "demo" del funcionamiento del indexador y del buscador. Se pueden incluir capturas de pantalla:

py ./exec.py

Hemos realizado un pequeño *script* "*exec.py*" que realiza la ejecución de los distintos programas, de forma que ejecuta *SAR_indexer.py* para la creación del archivo *.bin*, que es el que emplea *SAR_Searcher.py y* después ejecuta este, indicándoles las distintas palabras a comprobar y los distintos *threshold* a probar.

Aquí podemos apreciar una imagen de nuestro *script*:

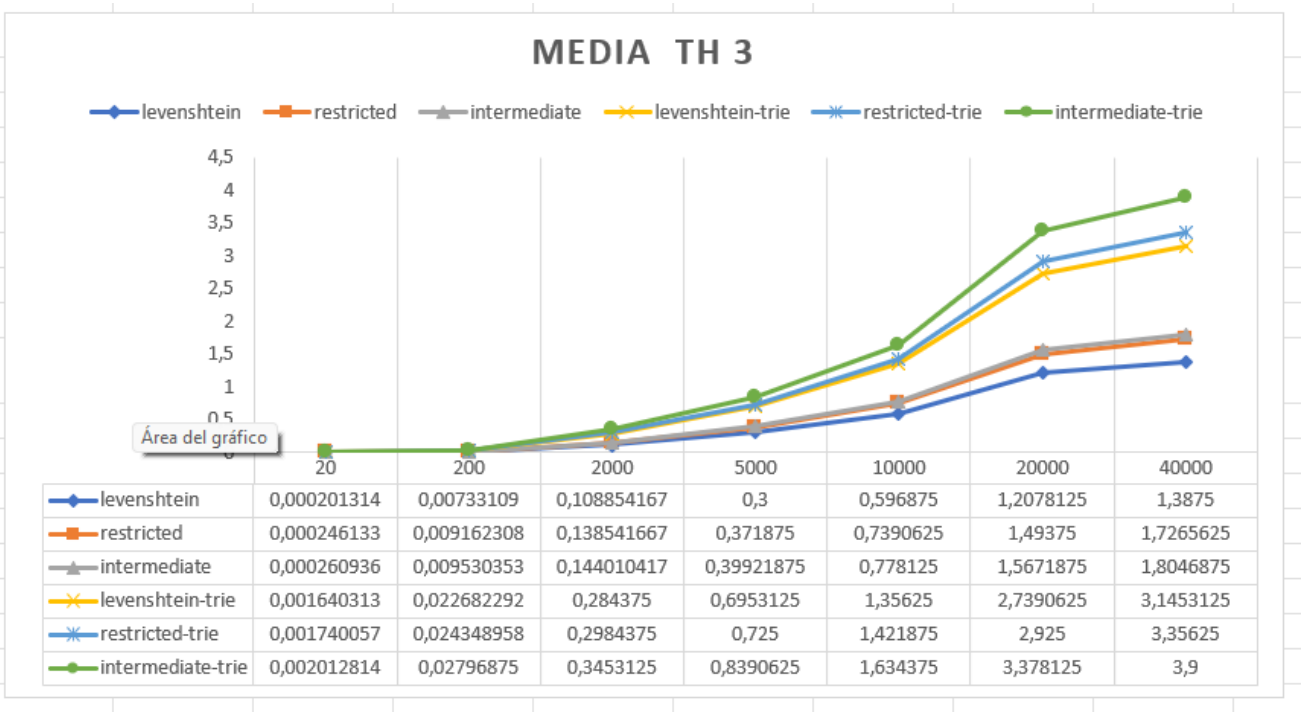
```
import os

os.system('py .\SAR_Indexer.py -S -P -M -O -U corpora/2015 2015_index_full.bin')
for i in range(1, 4):
    for dis in ["levenshtein"]:
        print("TH: %d" % i)
        print(dis)
        for word in ["alexanderx", "iluminaciónx", "capitla", "secuelax",
                     "asoman2", "virneg", "perwisos", "persimos", "limertad", "sagardo",
                     "aperturaq", "loursed", "cuaqro", "planrta", "NOT antihéroes AND ecuador",
                     "blindarse OR árboles AND perjuicio", "orígenes AND aversión",
                     "actualmente AND NOT reubicar", "ventanasse OR rebuscar ",
                     "tortuosa OR atardecer", "NOT alexander AND iluminación",
                     "capital OR secuela AND asoman"]:
            os.system('py .\SAR_Searcher.py -C -Q "' + word +
                      '" -E -D ' + dis + ' -H ' + str(i) + ' 2015_index_full.bin')
```

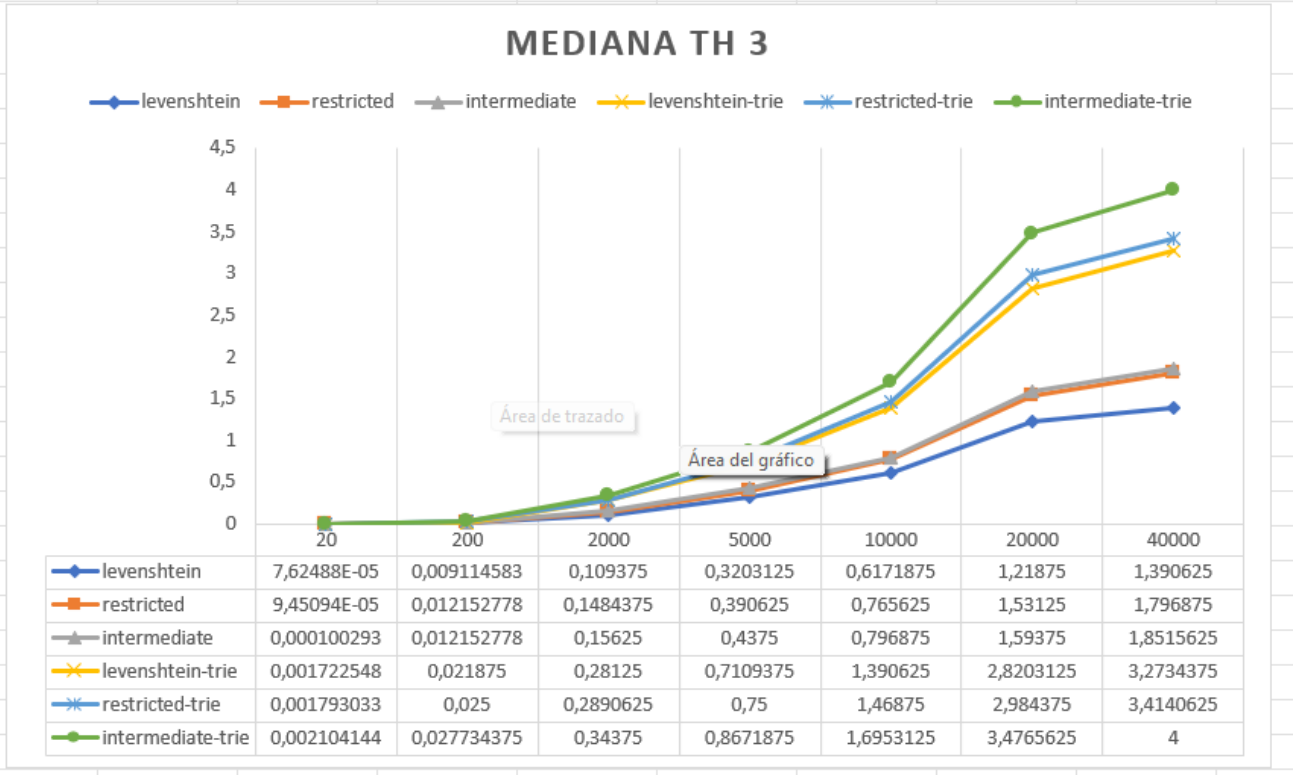
Obteniendo, así, los resultados de la ejecución de la búsqueda aproximada en nuestro buscador de noticias, que serán los que compararemos con los esperados para estos algoritmos y con los *threshold* indicados, viendo cómo coinciden con los proporcionados en *PoliformaT*. A continuación, mostraremos una imagen de la salida de este para una pequeña cantidad de valores:

```
TH: 1
levenshtein
alexanderx      2
iluminaciónx    8
capitla 10
secuelax        2
asoman2 2
virneg  0
perwisos        6
persimos        1
limertad        58
sagardo 0
aperturaq       26
loursed 0
cuaqro  184
planrta 50
NOT antihéroes AND ecuador      11
blindarse OR árboles AND perjuicio      0
orígenes AND aversión   1
actualmente AND NOT reubicar    73
ventanasse OR rebuscar  2
tortuosa OR atardecer   9
NOT alexander AND iluminación   8
capital OR secuela AND asoman   0
TH: 2
```

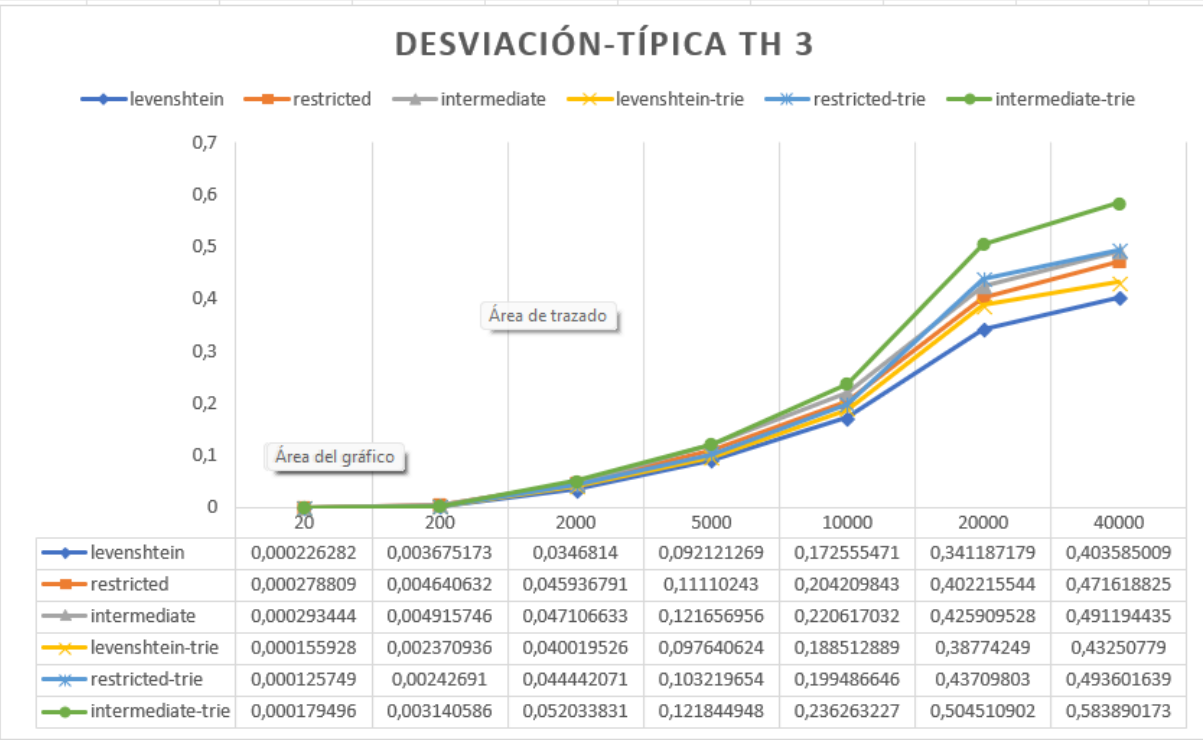
Media, mediana y desviación típica para un *threshold* de 3



	20	200	2000	5000	10000	20000	40000
levenshtein	0,000201314	0,00733109	0,108854167	0,3	0,596875	1,2078125	1,3875
restricted	0,000246133	0,009162308	0,138541667	0,371875	0,7390625	1,49375	1,7265625
intermediate	0,000260936	0,009530353	0,144010417	0,39921875	0,778125	1,5671875	1,8046875
levenshtein-trie	0,001640313	0,022682292	0,284375	0,6953125	1,35625	2,7390625	3,1453125
restricted-trie	0,001740057	0,024348958	0,2984375	0,725	1,421875	2,925	3,35625
intermediate-trie	0,002012814	0,02796875	0,3453125	0,8390625	1,634375	3,378125	3,9

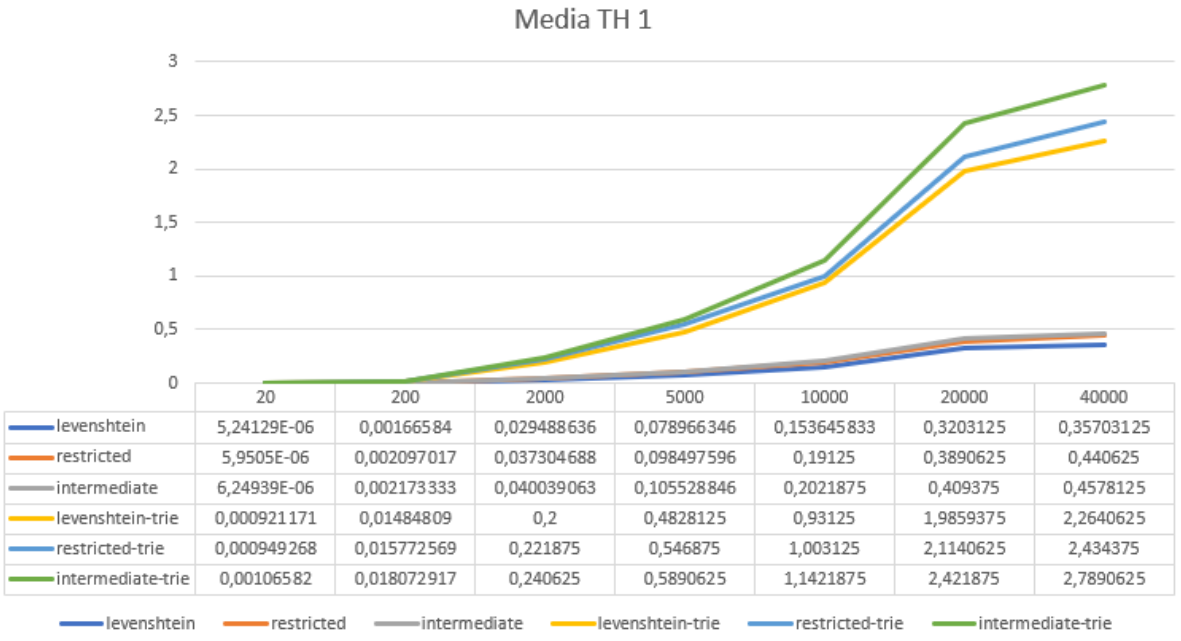


	20	200	2000	5000	10000	20000	40000
levenshtein	7,62488E-05	0,009114583	0,109375	0,3203125	0,6171875	1,21875	1,390625
restricted	9,45094E-05	0,012152778	0,1484375	0,390625	0,765625	1,53125	1,796875
intermediate	0,000100293	0,012152778	0,15625	0,4375	0,796875	1,59375	1,8515625
levenshtein-trie	0,001722548	0,021875	0,28125	0,7109375	1,390625	2,8203125	3,2734375
restricted-trie	0,001793033	0,025	0,2890625	0,75	1,46875	2,984375	3,4140625
intermediate-trie	0,002104144	0,027734375	0,34375	0,8671875	1,6953125	3,4765625	4

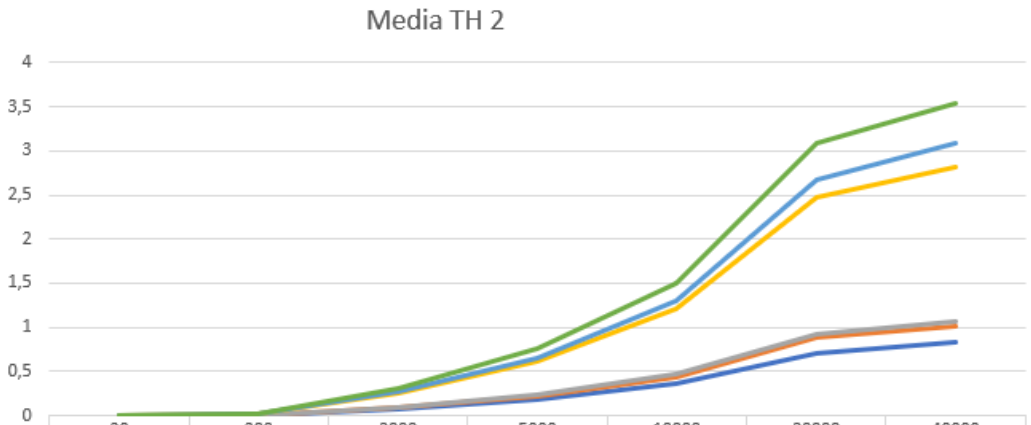


	20	200	2000	5000	10000	20000	40000
levenshtein	0,000226282	0,003675173	0,0346814	0,092121269	0,172555471	0,341187179	0,403585009
restricted	0,000278809	0,004640632	0,045936791	0,11110243	0,204209843	0,402215544	0,471618825
intermediate	0,000293444	0,004915746	0,047106633	0,121656956	0,220617032	0,425909528	0,491194435
levenshtein-trie	0,000155928	0,002370936	0,040019526	0,097640624	0,188512889	0,38774249	0,43250779
restricted-trie	0,000125749	0,00242691	0,044442071	0,103219654	0,199486646	0,43709803	0,493601639
intermediate-trie	0,000179496	0,003140586	0,052033831	0,121844948	0,236263227	0,504510902	0,583890173

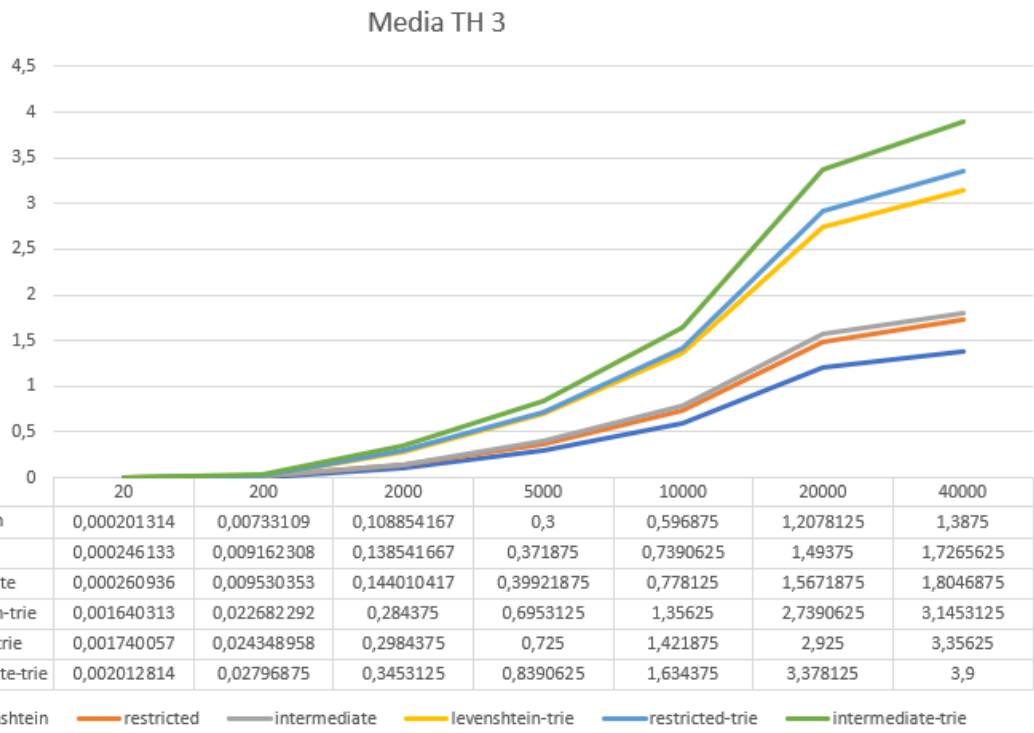
Media de los distintos algoritmos



	20	200	2000	5000	10000	20000	40000
levenshtein	5,24129E-06	0,00166584	0,029488636	0,078966346	0,153645833	0,3203125	0,35703125
restricted	5,9505E-06	0,002097017	0,037304688	0,098497596	0,19125	0,3890625	0,440625
intermediate	6,24939E-06	0,002173333	0,040039063	0,105528846	0,2021875	0,409375	0,4578125
levenshtein-trie	0,000921171	0,01484809	0,2	0,4828125	0,93125	1,9859375	2,2640625
restricted-trie	0,000949268	0,015772569	0,221875	0,546875	1,003125	2,1140625	2,434375
intermediate-trie	0,00106582	0,018072917	0,240625	0,5890625	1,1421875	2,421875	2,7890625

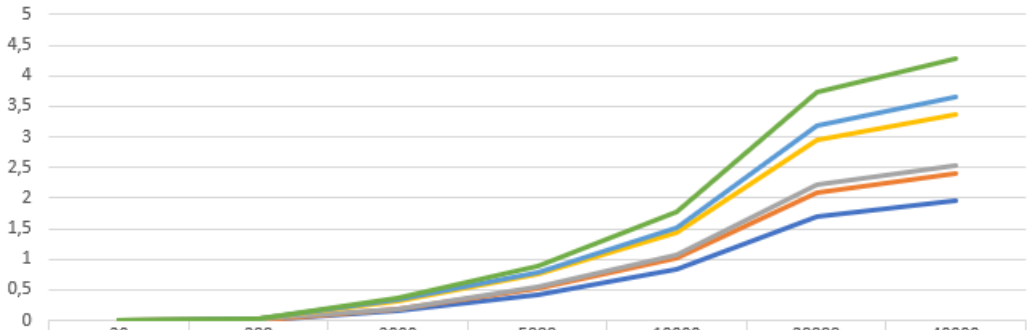


	20	200	2000	5000	10000	20000	40000
levenshtein	5,06822E-05	0,004022543	0,065058877	0,1865625	0,36328125	0,7109375	0,8234375
restricted	6,40791E-05	0,005308967	0,082057292	0,223046875	0,4421875	0,8828125	1,009375
intermediate	6,86238E-05	0,005545841	0,087847222	0,2359375	0,4640625	0,9296875	1,0734375
levenshtein-trie	0,001334582	0,019233631	0,2484375	0,615625	1,203125	2,46875	2,8140625
restricted-trie	0,001405915	0,022734375	0,2734375	0,6546875	1,296875	2,6796875	3,0859375
intermediate-trie	0,001608279	0,024817708	0,3046875	0,7546875	1,49375	3,0890625	3,540625



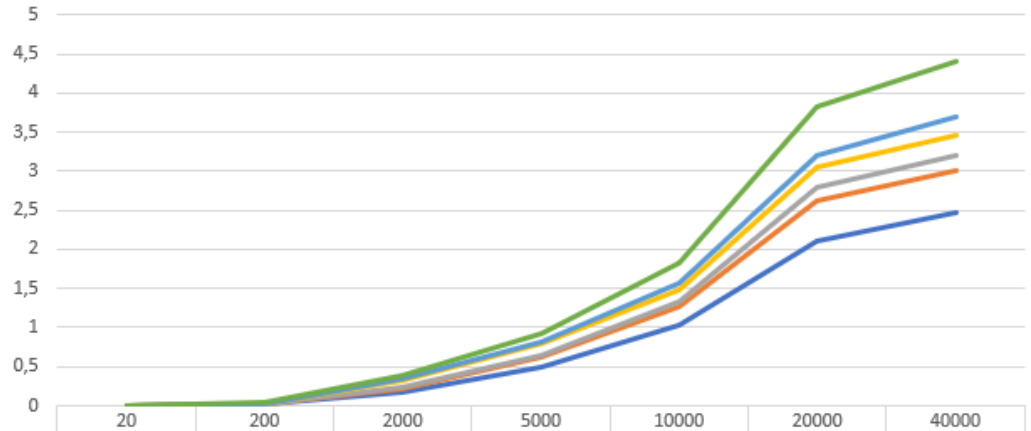
	20	200	2000	5000	10000	20000	40000
levenshtein	0,000201314	0,00733109	0,108854167	0,3	0,596875	1,2078125	1,3875
restricted	0,000246133	0,009162308	0,138541667	0,371875	0,7390625	1,49375	1,7265625
intermediate	0,000260936	0,009530353	0,144010417	0,39921875	0,778125	1,5671875	1,8046875
levenshtein-trie	0,001640313	0,022682292	0,284375	0,6953125	1,35625	2,7390625	3,1453125
restricted-trie	0,001740057	0,024348958	0,2984375	0,725	1,421875	2,925	3,35625
intermediate-trie	0,002012814	0,02796875	0,3453125	0,8390625	1,634375	3,378125	3,9

Media TH 4



	20	200	2000	5000	10000	20000	40000
levenshtein	0,000383685	0,010347283	0,149479167	0,40625	0,825	1,6984375	1,953125
restricted	0,000484588	0,012870555	0,18515625	0,5328125	1,021875	2,0859375	2,4046875
intermediate	0,000534041	0,013328534	0,194270833	0,5421875	1,0796875	2,2109375	2,5421875
levenshtein-trie	0,001898625	0,025546875	0,30625	0,7546875	1,4390625	2,9390625	3,378125
restricted-trie	0,001994646	0,026875	0,3265625	0,7703125	1,5265625	3,184375	3,6453125
intermediate-trie	0,002300228	0,03109375	0,3609375	0,884375	1,7796875	3,7421875	4,2859375

Media TH 5



	20	200	2000	5000	10000	20000	40000
levenshtein	0,000547474	0,01225828	0,1765625	0,496875	1,025	2,115625	2,4671875
restricted	0,000647116	0,014922664	0,221875	0,621875	1,2578125	2,6171875	3,01875
intermediate	0,000688288	0,015010626	0,2328125	0,653125	1,325	2,784375	3,2125
levenshtein-trie	0,002005897	0,026822917	0,3234375	0,7921875	1,484375	3,0453125	3,4640625
restricted-trie	0,00216601	0,027682292	0,3390625	0,8078125	1,5640625	3,2125	3,7
intermediate-trie	0,002485434	0,033697917	0,38125	0,921875	1,828125	3,8375	4,403125