Índice:

0.- Introducción

1.- Levenshtein y Levensthtein-Damerau de cadenas

2.- Trie con PD

3.- Levenshtein de cadena y trie con PD

4.- Levenshtein de cadena y trie con ramificación

5.- Adaptación código SAR

0.- Introducción

En este informe vamos a presentar el trabajo que hemos realizado según la programación de prácticas para adaptar el buscador de documentos de SAR para realizar búsquedas aproximadas de cadenas con tolerancia.

1- Levenshtein y Levensthtein-Damerau de cadenas

Para realizarlo hemos hecho un método auxiliar para crear la estructura de datos (matriz) e inicializarla según la distancia de Levenshtein. Esto se realiza en el método ini\_distancia.

2.- Trie con PD

Para realizar el Trie, hemos usado una estructura tipo lista de nodos (clase Node), y otra lista auxiliar que contiene los nodos finales (que representan una palabra completa).

El principal método que hemos implementado es add\_son, donde se le da una palabra y, tras una búsqueda previa en el trie, la añade si la palabra no está, se completa si está en parte o no se añade si ya esta en el Trie.

3.- Levenshtein de cadena y trie con PD

La función que implementa la distancia de Levenshtein usando como estructura de datos para la búsqueda un Trie esta situada en SAR\_library con el nombre levesteinTree\_Word\_PD. En él, se inicializa la estructura de datos tipo matriz a ceros y se va completando con las distancias de levenshtein mínimas para cada letra de la palabra.

4.- Levenshtein de cadena y trie con ramificación

En la versión con ramificación, se implementa una cola FIFO cuyos elementos son tuplas de tres elementos (Indice de la letra dentro de la palabra que se busca en el Trie, Indice del nodo de la letra que se encuentra en el Trie, Coste de la operación de edición).

La versión de Levenshtein tiene en cuenta el borrado, la inserción y la sustitución de letras de la cadena y en la versión Levenhtein-Damerau su añade la operación de intercambio entre letras contiguas de la misma cadena.

5.- Adaptación código SAR

Se ha adaptado el código para que admita la búsqueda con tolerancia. En caso de usar la distancia de Levenshtein, se usará el carácter % para indicar la tolerancia en la búsqueda y en el caso de de querer usar la distancia de Lev-Damerau se usará el carácter @.

Teniendo en cuenta que el coste de el algoritmo de programación dinámica en este caso es constante O(k) y el de ramificación es exponencial O(xn), se usa el primero para los casos donde la tolerancia sea un numero alto (3 o más) y para los casos con una tolerancia menor (1 o 2) se utiliza el de ramificación.

Cambien se ha adaptado el código para que sea posible hacer búsquedas usando puertas lógicas como or, and y not; con la búsqueda con tolerancia.

Comparación de tiempos entre los algoritmos de programación dinámica y ramificación.

Tabla de tiempos del algoritmo de la distancia de Levenshtein:

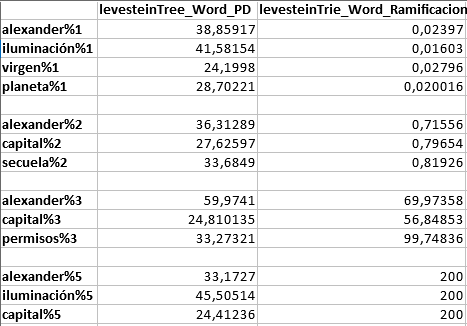


Imagen 1: En el caso de ramificación, se ha optado por poner un numero mucho mas alto que el resto dado que la ejecución del algoritmo suponía un tiempo excesivamente alto.

Los valores usados en la grafica son las medias de la Imagen 1, que a continuación se muestran:

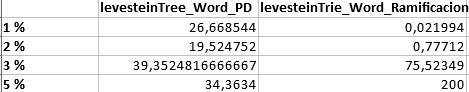


Imagen 2

Tabla de tiempos del algoritmo de la distancia de Damerau-Levenshtein:

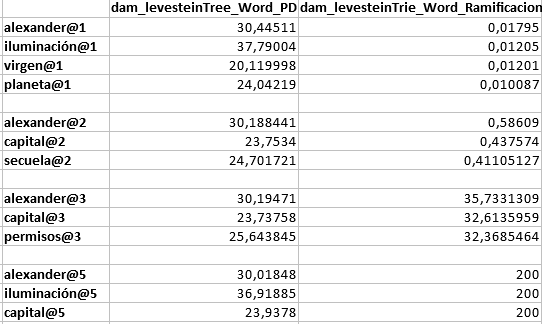
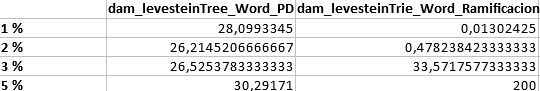
Imagen 3

Imagen 4

Imagen 5