TP - Unidad 02
Algoritmos de Búsqueda en Texto Aproximada- Parte A

Ejercicio 1

1.1) Aplicar el algoritmo fonético Soundex a cada una de las siguientes palabras (pasar todo mayúsculas antes de hallar el encoding):

brooklin, bruqleen, brooclean, bluclean, clean

1.2) Calcular la similitud soundex entre brooklin y los demás 4 strings.

Ejercicio 2

2.1) Implementar en Java el algoritmo Soundex. Realizar los testeos de unidad para verificar correctitud.

Ej:
new Soundex ("threshold") debe devolver el String "T624"
new Soundex ("hold") debe devolver el String "H430"
new Soundex ("phone") debe devolver el String "P500"
new Soundex ("foun") debe devolver el String "F500"

2.2) Calcular la complejidad temporal y espacial del algoritmo propuesto en el ítem 2.1)

2.3) Implementar en Java el algoritmo de similitud entre dos strings, a partir de encoding Soundex. Agregar nuevos testeos de unidad.

Caso de uso:
double s1= Soundex.similarity(“threshold”, “hold”) debe devolver el double 0.0
s1= new Soundex("phone ").similarity("foun") debe devolver el double 0.75

2.4) Explicar cuáles son los únicos valores posibles de similitud que se puede obtener con Soundex.

Ejercicio 3

3.1) Aplicar el algoritmo fonético ***Metaphone*** a cada una de las siguientes palabras (pasar todo mayúsculas antes de hallar en encoding):

brooklin, bruqleen, brooclean, bluclean, clean

3.2) Calcular la similitud metaphone entre brooklin y los demás 4 strings. ¿Mejoró alguna respecto de la similitud soundex? Explicar. (Para calcular la similitud hay formas interesantes de calcularlas como veremos luego…)

3.3) Proponer un par de strings donde Metaphone da mejor similitud que Soundex debido a que usa una o varias de las 19 reglas. Explicar además cuáles reglas fueron usadas para la propuesta según https://en.wikipedia.org/wiki/Metaphone y por eso su mejora.

Ejercicio 4

Para cada uno de los siguientes pares de strings:

brooklin y bruqleen
brooklin y brooclean
brooklin y bluclean
brooklin y clean

4.1) Calcular la distancia de Levenshtein exhibiendo la matriz que permite calcular la distancia por medio de la aplicación de la técnica de programación dinámica (algoritmo propuesto por Robert A. Wagner and Michael J. Fischer)

4.2) Calcular la similitud normalizada entre cada par de strings.

Ejercicio 5

En general, usando similitud con Levenshtein no dio mejor que soundex o metaphone porque las palabras a comparar eran similares fonéticamente y no por errores de edición. Proponer un par de Strings donde la similitud por Levenshtein da mejor que con soundex y metaphone. Justificar la respuesta.

Ejercicio 6

6.1) Implementar en Java el cálculo de la distancia de Levenshtein (algoritmo propuesto por Robert A. Wagner and Michael J. Fischer) y la similitud normalizada. Realizar testeos de unidad para verificar correctitud.

Caso de uso:

int dist= Levenshtein.distance("big data", "bigdaa"); // debería devolver el entero 2

double simil= Levenshtein.normalizedSimilarity("big data", "bigdaa"); //deberia devolver el double 0.75

6.2) Calcular la complejidad temporal y espacial del algoritmo propuesto en 6.1)

Ejercicio 7

7.1) Escribir un algoritmo en Java para el cálculo de la distancia de Levenshtein que use complejidad espacial O(min(s1.length, s2.length)). Deben seguir resultando exitosos los testeos de unidad anteriores.

7.2) Calcular la complejidad temporal del algoritmo propuesto en 7.1)

Ejercicio 8

8.1) Calcular los Q-Grams donde Q es exactamente 2, para cada uno de los siguientes strings: "salesal" Y "alale"

8.2) Calcular la similitud de ambos strings para Q exactamente 2.

Ejercicio 9

En la práctica, por simplicidad, muchas veces se usa Q exactamente 3. Proponer un par de strings donde la similitud con Q-gram para Q=3 da mejor que la similitud de Levenshtein. Explicar por qué la mejora.

Ejercicio 10

10.1) Escribir la clase Java QGram que calcule los Q-grams para Q exactamente N y N es un parámetro del constructor. Dicha clase deberá además tener dos métodos de instancia :

- printTokens(String) para obtener los pares Qgram y cantidad de apariciones
- similarity(String, String) que calcula la similitud normalizada entre ambos strings.,

Realizar testeos de unidad para verificar correctitud.

Caso de Uso:

```
QGram g= new Qgram(2);  // 1, 2, 3 .etc
g.printTokens("alal");              // no importa el orden del output
  //  #a  1
  //  al  2
  //  la  1
  //  l#  1
…
double  value= g.similarity("salesal", "alale");
System.out.println(value);  //
…
```

10.2) ¿Qué ventajas o desventajas tiene realizar la implementación con ArrayList o HashMap?

Ejercicio 11

11.1) Analizar si la clase String de Java tiene comparación de Strings aproximado (Soundex, Metaphone, Levenshtein, Q-Grams).

11.2) Existe una biblioteca denominada Apache Commons (https://commons.apache.org) que posee manejo avanzado de Strings. Crear un proyecto Maven y agregan las dependencias necesarias para manejar los algoritmos antes discutidos. Intentar hacer un wrapper sobre lo que tenga esta biblioteca para obtener como mínimo:

- Similitud entre 2 strings por medio de Soundex, Levenshtein y Q-grams según Apache Commons
- Donde aplique, agregar métodos para obtener la información máxima posible de cada algoritmo. Ej: para soundex agregar un método para obtener el encoding, en q-grams obtener lo análogo al printTokens, para Levenshtein la cantidad de sustituciones/inserciones y borrados, etc.

¿Hay algo que no pudieron implementar?

Verificar que lo obtenido en la implementación previa, coincide con lo que devuelve esta biblioteca.

11.3) Para el caso de Levenshtein averiguar qué clase permite obtener la cantidad de inserciones, borrados y sustituciones que hay que hacer para transformar un string en otro. Agregar dicha funcionalidad específicamente para Levenshtein.

Ejercicio 12

Dado que en Metaphone el encoding es de longitud variable, resulta difícil encontrar un valor adecuado de similitud cuando los encoding no miden lo mismo. Tal es el caso de la Similitud entre brooklin y clean. Una algoritmo de similitud interesante para Metaphone consiste en, una vez calculado el encoding, aplicar la similitud normalizada de Levenshtein sobre los encoding y que sea esa la métrica de similitud.

Aprovechando que Apache Commons tiene implementado Metaphone, agregar al proyecto del ejercicio anterior la similitud para Metaphone pero haciendo uso de esta estrategia. Realizar testeos de unidad. Verificar que la similitud entre brooklyn y clean mejora con esta estrategia (deberían obtener distancia 2, similitud 0.6)

Ejercicio 13

Analizar el código de cómo implementó Apache Commons las clases para Soundex, Metaphone y Levenshtein, y compararla con las implementaciones que Uds. propusieron previamente.

Ejercicio 14

Cuando estamos desarrollando un proyecto es común encontrarnos con que una biblioteca de terceros no incluye todas las funcionalidades que estamos necesitando. De ahí la importancia de usar Maven.

En este caso, el problema lo encontramos con Q-Grams.

14.1) Agregar al proyecto la dependencia java-string-similarity. La misma incluye la clase Qgram que, similar a lo que realizaron previamente, incluye parametrización en Q. Concatenar antes de invocar los símbolos # necesarios, ya que la biblioteca no lo hace automáticamente.

Usando esta biblioteca deberán calcular la similitud y printTokens() como antes. Chequear que los testeos de unidad permiten obtener lo mismo que antes.

14.2) Qué algoritmos que discutimos no tiene implementada esta biblioteca?

Ejercicio 15

Analizar el código de cómo implementó la biblioteca anterior a la clase y compararla con la implementación que Uds. propusieron previamente.

Ejercicio 16

Se tiene un e-commerce. El mismo ofrece ciertos productos. Los usuarios pueden libremente realizar búsqueda de productos en dicho sitio. Puede ser que el producto ingresado coincida con el almacenado o que por errores fonéticos o de edición no matcheen directamente. Sin embargo, en vez de devolver un resultset vacío, se prefiere devolver una lista con los 5 productos más parecidos a la búsqueda deseada.

Escribir una aplicación maven (no tiene por qué tener una interface gráfica) que permita ingresar una palabra y devuelva los 5 productos más similares, rankeados por similaridad.

Para ello hay que aplicar similitud con Soundex, Metaphone, Levenshtein y QGrams(=3) entre la palabra ingresada (input) y los productos que se posee. Por cada producto que se tiene en el e-commerce calcular las 4 similitudes con el input: soundex, metaphone, Levenshtein y QGrams(=3) y quedarse con la mejor (máximo valor y algoritmo/s). Ordenar globalmente todos los mejores resultados obtenidos previamente y mostrar no solamente los productos más parecidos sino el valor de similitud y el algoritmo que dio mejor.

Los productos que vende este e-commerce se encuentran en un archivo de texto. Dado que en el archivo los datos están procesados y no contienen más de

un espacio en blanco, están en minúsculas, etc procesar solo el input del usuario para maximizar la coincidencia.

- Cargar el catálogo desde el archivo product.txt (parametrizarlo por si se desea cambiar el archivo).
- Reglas mínimas Dominio Dependientes a aplicar:
    - Dejar solo un blanco como separador interno de palabras.
    - Pasa todo a minúscula.
    - Si ingresa acentos reemplazarlos por las letras sin acentos. La ñ por la n.

Ej : Si el usuario ingresa “ porta roLLo" deberá buscar “porta rollo”.

- Realizar testeo de unidad para probar como mínimo con siguientes palabras:

    - portarollo
    - secaplatos
    - tenderero
    - tender
    - ténder
    - porta royo
    - seca plato

No olvidar generar los testeos de unidad para verificar correctitud.