Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación
R-212 — Estructuras de Datos y Algoritmos I

Trabajo Práctico Final

Corrector Ortográfico

1. Motivación del problema

Muchas aplicaciones, incluyendo editores de texto, procesadores, y clientes de e-mail incluyen un corrector ortográfico. El trabajo de un corrector ortográfico es relativamente simple: dada una lista de palabras correctas (la cual llamaremos nuestro *diccionario*) y un texto de entrada; el corrector ortográfico debe detectar todos los errores de ortografía, es decir, todas las palabras del texto no encontradas en el diccionario.

Cuando se encuentra un error de ortografía, un buen corrector debería sugerir palabras en nuestro diccionario que sean correctas, y se *aproximen* a la palabra errónea.

Veamos un ejemplo, supongamos que en nuestro diccionario están las palabras:

- vaso,
- caso,
- bazo,
- brazo,
- craso,

y recibimos una entrada que contiene la palabra `vazo`. Dado que no está en el diccionario de palabras, se supone que es entonces un error de ortografía. La tarea del corrector es sugerir palabras del diccionario que son cercanas a la recibida, por ejemplo: vaso o caso o bazo.

A diferencia del ejemplo anterior, la tarea de un corrector ortográfico se centra en buscar palabras en un diccionario grande. Una búsqueda eficiente es crítica para la performance del corrector ortográfico, dado que buscar en el diccionario no implica sólo buscar cada palabra de la entrada sino posiblemente, cientos de sugerencias para cada error ortográfico que se encuentre.

2. Objetivos

El programa que se tendrá que realizar, será un corrector ortográfico rudimentario: éste leerá un archivo de diccionario (provisto por la Cátedra), tomará un archivo de texto de entrada y, por cada error ortográfico encontrado, el programa deberá entregar una lista de sugerencias (si es que hay alguna).

3. Detalles de la implementación

Existen dos correctores ortográficos populares en Unix/Linux. Uno se llama **ispell** (`https://www.cs.hmc.edu/~geoff/ispell.html`), el otro es un un programa llamado **aspell** (`http://aspell.net/`) que tiene licencia GNU. Ambos usan técnicas similares para sugerir palabras. Mientras están chequeando la ortografía de un archivo de entrada, si una palabra no está presente en el diccionario, usan 5 técnicas para

generar posibles sugerencias. Cada posible sugerencia es buscada en el diccionario; si es encontrada se agrega como sugerencia. Las cinco técnicas usadas son:

- Intercambiar cada par de caracteres adyacentes en la palabra.
- Insertar cada letra de la 'A' a la 'Z' en cualquier posición de la palabra (al principio, entre dos caracteres, o al final).
- Eliminar cada caracter de la palabra.
- Reemplazar cada caracter de la palabra con cada letra de la 'A' a la 'Z'.
- Separar la palabra en un par de palabras agregando un espacio entre cada par de caracteres adyacentes en la palabra. En este caso, se debería sugerir si ambas palabras pertecen al diccionario.

El programa a entregar deberá generar sugerencias usando estas cinco técnicas y mostrarlas al usuario. Las sugerencias deben darse *en orden de distancia*: si una sugerencia requiere un sólo paso (e.g. "casa" desde "kasa") debe mostrarse antes que las sugerencias a las que se llega en dos pasos (e.g. "pala"). No hay orden definido para palabras que estén a la misma distancia (*empatadas*), cualquier orden entre ellas está bien. Es decir, ninguna técnica/regla tiene prioridad sobre otra.

El corrector deberá generar hasta cinco sugerencias para cada palabra mal escrita. El corrector *nunca* debe retornar más de cinco sugerencias. Si al agotar los 3 pasos (ver abajo), no se llega a cinco sugerencias, simplemente se deben mostrar las que se hayan encontrado (si es que hay alguna).

El programa debería funcionar con sugerencias que requieran hasta 3 pasos de las técnicas anteriores. Es decir, si hay sugerencias a 3 pasos o menos de la palabra original, deben poder encontrarse y mostrarse, salvo que ya se haya llegado al máximo de cinco sugerencias.

Hay una excepción para la última regla, la cuál sólo aplica si al separar la palabra conseguimos dos palabras que *ya están* en el diccionario, y no admite pasos posteriores. En otras palabras, al separar una cadena *nunca* se continúan aplicando pasos. Si al separar la cadena ambas subcadenas son palabras correctas, se genera *una* nueva sugerencia, y no hay efecto en caso contrario.

Entonces, volviendo a nuestro ejemplo de diccionario, para la palabra `eo` el programa podría sugerir `vaso`: 1) insertar una `v` antes de la `e`, 2) insertar una `s` entre la `e` y la `o`, 3) reemplazar la `e` por una `a`. Por el contrario, para la palabra `zzzzz` no existen en nuestro diccionario sugerencias a una distancia de 3 pasos.

El método main debería permitir pasar el nombre del archivo de entrada, el nombre del archivo de salida. Es decir, sería algo asi:

```
main archivoEntrada archivoSalida
```

3.1. Formato de archivo de entrada

El archivo de entrada puede tener múltiples líneas de texto. Cada una de ellas pueden tener varias palabras, las cuales estarán separadas por un solo espacio; también pueden estar los siguientes carácteres especiales inmediatamente a continuación de una palabra, los cuales deben ser ignorados: ":", ";", ",", ".", "?", "!".

También, en el archivo de entrada podrían existir líneas vacías. Considere que las palabras podrían estar escritas con mayúsculas, minúsculas, o una mezcla de ambas.

3.2. Formato de archivo de salida

Por cada palabra no incluida en nuestro diccionario que encontremos en el archivo de entrada, se deberá imprimir un bloque como se indica a continuación. Al final de cada bloque se deberá imprimir una línea vacía para separar. El bloque debe decir:

`Linea N, "PALABRA" no esta en el diccionario.`
`Quizas quiso decir: SUGERENCIA`$_1$`, SUGERENCIA`$_2$`, ..., SUGERENCIA`$_N$

3.3. Adicional para mesas de diciembre

Además de los archivos con el universo y el texto de entrada, el programa deberá leer un archivo adicional con sugerencias calculadas previamente. Cada línea del archivo de sugerencias de entrada tendrá el siguiente formato:

`palabra, nro_sugerencias, sugerencia`$_1$`, ..., sugerencia`$_N$

Es decir, una palabra no perteneciente al universo, seguida por el número de sugerencias que vendrán a continuación y finalmente una lista con cada una de las palabras sugeridas.

La información de este archivo deberá ser utilizada para optimizar el funcionamiento del programa. Por ejemplo, si una palabra del texto de entrada ya se encuentra en este archivo, entonces se deberá tomar como sugerencias a las que aparecen listadas. De lo contrario, el programa procederá de la manera habitual, es decir, generará las sugerencias mediante la aplicación de las técnicas mencionadas.

Al finalizar la ejecución, el programa agregará a este archivo las nuevas sugerencias generadas, para que puedan ser aprovechadas en las próximas ejecuciones.

4. Evaluación

Para la evaluación del Trabajo Práctico se tomarán en cuenta los siguientes elementos:

a) Estructuras de datos usadas.

b) Algoritmos propuesto.

c) Eficiencia.

d) Calidad del código entregado.