Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI2692 - Laboratorio de Algoritmos y Estructuras II
Trimestre Abril-Julio 2023

Proyecto Ayudante Ortográfico

El objetivo del proyecto es la implementación de un ayudante ortográfico. Esta herramienta carga un diccionario desde un archivo y además el ayudante debe ser capaz de revisar un texto y detectar las palabras que no se encuentren en el diccionario. A cada una de las estas palabras que no están en el diccionario, se les va a buscar las palabras que les sean más cercanas para presentarlas al usuario como recomendaciones.

1. Preliminares

1.1. Función palabra válida

Se tiene que un elemento de tipo String es una *palabra válida* si está formado por caracteres alfabéticos en minúsculas, esto es los caracteres entre 'a'y 'z', y agregando el carácter eñe ('ñ'). Un String, puede ser representado como una secuencia de caracteres. Para determinar si un String es una *palabra válida* definimos la función *esPalabraValida* como sigue:

$$esPalabraValida(s) \quad \equiv \quad (\forall\, i : 0 \leqslant i < len(s) : \text{`}\texttt{a}\text{'} \leqslant s[i] \leqslant \text{`}\texttt{z}\text{'} \ \vee \ s[i] = \text{`}\texttt{ñ}\text{'})$$

Se tiene que *len* una la función que retorna la longitud del elemento de tipo String y que $s[i]$ denota el i-ésimo carácter del String s.

1.2. Distancia entre palabras

Para poder determinar la similitud entre dos palabras en necesario contar con un algoritmo que mida la distancia entre ellas. Para medir la distancia entre dos Strings se debe utilizar una métrica de comparación de cadenas de caracteres. En específico, debe usar la métrica conocida como la distancia Damerau-Levenshtein, en específico la *optimal string alignment distance* [2].

2. Tipos de datos

En esta sección presentamos los TADs más relevantes para la realización del proyecto.

2.1. TAD `Palabras con la Misma Letra Inicial` (PMLI)

Este tipo de dato contiene a un conjunto de palabras, las cuales comienzan por una misma letra. A continuación se presenta la especificación del TAD.

Especificación del TAD PMLI

Modelo de Representación

var *letra* : Char

var *palabras* : Conjunto de Strings

Invariante de Representación

$(letra = \text{`a'} \ \vee \ letra = \text{`b'} \ \vee \ \ldots \ \vee \ letra = \text{`z'} \ \vee \ letra = \text{`ñ'}) \ \wedge$
$(\forall\, p)(p \in palabras \Rightarrow (p[0] = letra \ \wedge \ esPalabraValida(p)))$

Operaciones

fun *crearPMLI* (**in** l : Char) $\rightarrow$ *PMLI*
{ **Pre:** True }
{ **Post:** $letra = l \ \wedge \ palabras = \varnothing$ }
proc *agregarPalabra* (**in** p : String)
{ **Pre:** $p[0] = letra$ }
{ **Post:** $palabras = palabras_0 \cup \{p\}$ }
proc *eliminarPalabra* (**in** p : String)
{ **Pre:** $p[0] = letra$ }
{ **Post:** $palabras = palabras_0 - \{p\}$ }
proc *mostrarPalabras* ( )
{ **Pre:** True }
{ **Post:** Muestra por la salida estándar los elementos en *palabras* en orden lexicográfico. }
fun *buscarPalabra* (**in** p : String) $\rightarrow$ Boolean
{ **Pre:** $p[0] = letra$ }
{ **Post:** $buscarPalabra \equiv (p \in palabras)$ }

Fin TAD

2.2. TAD `Ayudante Ortográfico`

Este TAD posee como principal estructura de datos un arreglo de `PMLI` de tamaño 27. Cada casilla del arreglo corresponde a una letra del alfabeto latino, incluyendo al carácter 'ñ'. Esta estructura de datos es donde usted debe almacenar las palabras del diccionario o diccionarios que va a usar el TAD. A continuación se presenta el modelo de representación y el invariante del TAD.

Especificación del TAD Ayudante Ortográfico

Modelo de Representación

const MAX : int

var *dicc* : arreglo de PMLI

Invariante de Representación

$MAX = 27 \ \wedge \ \#dicc = 27 \ \wedge$
$dicc[0].letra = \text{`a'} \wedge dicc[1].letra = \text{`b'} \wedge \ldots \wedge dicc[26].letra = \text{`z'}$

Operaciones

...

Fin TAD

A continuación se describen las operaciones operaciones del TAD Ayudante Ortográfico.

2.2.1. Procedimiento `crearAyudante`

Crea un nuevo TAD Ayudante Ortográfico en donde la estructura de datos *dicc* se inicializa, generando un arreglo, donde se crean 27 instancias del TAD `PMLI`, una por cada letra del alfabeto.

fun *crearAyudante* () → *AyudanteOrtografico*
{ **Pre:** True }
{ **Post:** Queda inicializa la estructura *dicc*
creando las 27 instancias de TAD `PMLI`, una para cada letra del alfabeto. }

2.2.2. Procedimiento `cargarDiccionario`

Este procedimiento lee un archivo de texto que contiene las palabras de un diccionario, las cuales van a ser almacenadas en la estructura de datos *dicc*. La entrada del procedimiento es el nombre del archivo con las palabras del diccionario. El formato del archivo de diccionario es el siguiente: solo debe contener una palabra por línea y todas las palabras deben cumplir la definición de *esPalabraValida*. Después de finalizar la lectura del archivo y de haber verificado que el mismo tiene un formato válido, las palabras son cargadas en el estructura *dicc*, siguiendo las especificaciones del TAD `PMLI`. Observe que con este procedimiento es posible cargar varios archivos con diccionarios.

proc *cargarDiccionario* (**in** *fname* : String)
{ **Pre:** El archivo *fname* debe cumplir con el formato preestablecido. }
{ **Post:** Quedan agregadas en *dicc* las palabras de *fname*
cumpliendo la especificación del TAD `PMLI`. }

2.2.3. Procedimiento `borrarPalabra`

Este procedimiento recibe como entrada una palabra, si la misma se encuentra en *dicc* la elimina. Si la palabra no se encuentra en *dicc* la estructura no sufre ninguna modificación.

proc *borrarPalabra* (**in** p : String)
{ **Pre:** $esPalabraValida(p)$ }
{ **Post:** $dicc[x].palabras = dicc'[x].palabras$, donde $dicc'[x].palabras$ es el conjunto resultante de aplicar $dicc_0[x].eliminarPalabra(p)$ en la instancia $dicc_0[x]$ del TAD PMLI, que cumple el invariante de representación $dicc_0[x].letra = p[0]$, para $0 \leq x \leq 26$ }

2.2.4. Procedimiento `corregirTexto`

Este procedimiento recibe como entrada un archivo de texto con palabras a revisar. Este archivo además de palabras válidas, puede contener palabras inválidas, espacios en blanco, saltos de línea, espacio de tabulaciones y signos de puntuación [1]. El procedimiento `corregirTexto` **debe procesar el archivo de entrada para extraer únicamente las palabras válidas que contenga**, ignorando todos los demás elementos mencionados anteriormente. Luego debe determinar cuales son las palabras válidas que no es encuentran en el diccionario. A continuación, para cada una de estas palabras, debe calcular las cuatro palabras en el diccionario de palabras (estructura *dicc*), con menor valor de distancia Damerau-Levenshtein, con respecto a esta palabra. Las cuatro palabras con menor distancia, serán las palabras va a sugerir el corrector ortográfico. Finalmente, debe imprimir los resultados en un archivo de salida. El archivo de salida debe tener el siguiente formato. Cada palabra válida que no se encuentre en el diccionario debe comenzar una línea y luego le siguen las cuatros palabras con menor distancia, separadas por coma.

proc *corregirTexto* (**in** *finput* : String ; **in-out** *foutput* : String)
{ **Pre:** *finput* es un archivo de texto válido }
{ **Post:** Imprime en el archivo *foutput* cada una de las palabras válidas contenidas en el archivo *finput* que no se encuentren en *dicc*, seguidas de las cuatro palabras con menor distancia. }

2.2.5. Procedimiento `imprimirDiccionario`

Imprime por la salida estándar la estructura *dicc*, en un formato que permita entender fácilmente el contendido de la misma.

proc *imprimirDiccionario* ()
{ **Pre:** True }
{ **Post:** Imprime *dicc* por la salida estándar mostrando las palabras en orden lexicográfico }

3. Programa cliente

Debe implementar una aplicación cliente llamada `PruebaOrtografia.kt`, que al ejecutarla, proporciona al usuario de un menú simple que permite llevar a cabo todas las operaciones del TAD Ayudante Ortográfico. Al iniciar el cliente, entrará en una iteración de menú con las siguiente 6 opciones:

1. Crear un nuevo ayudante ortográfico.
2. Cargar un diccionario.
3. Eliminar palabra.
4. Corregir texto.
5. Mostrar diccionario.
6. Salir de la aplicación.

Este menú se mostrará por la salida estándar y el usuario terminará la ejecución del cliente cuando seleccione la opción 6 "`Salir de la aplicación`". Si alguna de las precondiciones de los procedimientos de los TADs no se cumple, entonces se le debe indicar al usuario que la operación no pudo ser efectuada porque no se cumple la precondición, explicando cuál es esa precondición.

4. Requerimientos de la implementación

La implementación concreta del Conjunto de Strings del TAD PMLI, llamado *palabras*, debe ser hecha como una *tabla de hash* que resuelve colisiones por el método de encadenamiento. La función de hash, que retorna un entero, dado un String, a utilizar es el método `hashCode` de la clase String de Kotlin. Cuando al insertar un elemento en la *tabla de hash*, se tiene un factor de carga igual o mayor a 0.7, entonces se aplica la operación *rehashing*, en donde se duplica el tamaño de la tabla. Este TAD Conjunto debe ser implementado en una clase llamada `ConjuntoPalabras`, contenida en archivo llamado `ConjuntoPalabras.kt`.

La estructura *dicc* del TAD Ayudante Ortográfico, debe ser implementada como un arreglo. Una vez creada la estructura *dicc*, se deben agregar las 27 instancias del TAD PMLI. Luego esta estructura no cambia de tamaño.

Cada uno de los TADs debe ser implementado como una clase de `Kotlin`, contenida en un archivo del mismo nombre de la clase. Debe hacer entrega de por lo menos seis archivos:

- `PMLI.kt`: Implementación del TAD `PMLI`.

- `AyudanteOrtografico.kt`: Implementación del TAD `Ayudante Ortográfico`.
- `ConjuntoPalabras.kt`: TAD Conjunto implementado como una Tabla se *hash*.
- `PruebaOrtografia.kt`: Cliente que permite interactuar con el TAD `Ayudante Ortográfico`.
- `Makefile`: Archivo que compila todos los archivos del proyecto.
- `runPruebaOrtografia.sh`: Archivo *shell script* que ejecuta el cliente `PruebaOrtografia.kt`.

Como los TADs se deben a implementar como clases de `Kotlin`, la función `crearPMLI` del TAD `PMLI` va a corresponder al constructor de la clase, por lo no debe ser implementado como un método aparte. De la misma forma la función `crearAyudante` corresponde al constructor de la clase que implementa al TAD `Ayudante Ortográfico`, por lo que no debe ser creada como un método nuevo.

El código debe estar debidamente documentado y cada uno de los métodos de los TADs deben tener los siguientes elementos: descripción, descripción de los parámetros, precondición y postcondición. Además debe hacer uso de la guía de estilo de `Kotlin`.

5. Condiciones de entrega

Debe entregar los códigos fuentes de su programa, y la declaración de autenticidad, en un archivo comprimido llamado `Proyecto2-X-Y.tar.xz` donde X y Y son los números de carné de los autores del proyecto. La entrega debe hacerse por medio de la plataforma Classroom, antes de las 11:59 PM del día domingo 16 de julio de 2023.

Referencias

[1] Colaboradores de Wikipedia. Signos de puntuación — Wikipedia, la enciclopedia libre, 2021. [Online; revisado el 25-marzo-2021].

[2] Colaboradores de Wikipedia. Damerau-levenshtein distance — Wikipedia, la enciclopedia libre, 2023. [Online; revisado el 01-julio-2023].

Guillermo Palma / gvpalma@usb.ve / Julio 2023