UNIVERSIDAD DE BUENOS AIRES
Facultad de Ingeniería

Departamento de Computación

TRABAJO PRÁCTICO Nº 3

Primera Entrega – 6/12/2008

Algoritmos y Programación II (75.41)

Curso: 2- Patricia Calvo – Segundo Cuatrimestre de 2008

Día: Lu - Horario: 19 – 22 hs.

Día: Vi - Horario: 18:30 – 21:30 hs.

Profesores:

Ezequiel Soto.
Federico Zaiatz.
Gustavo Schmidt.
Mariano Simone.
Mariano Tugnarelli.
Patricia Calvo.

Ayudante Asignado:

Mariano Simone.

Integrantes del Grupo 16:

Apellido y Nombres	Padrón	Dirección de Correo Electrónico
Juaniquina Choque, Rolando.		
Ponce, Matias Ezequiel.		
Rinaldi, Lautaro Ezequiel.		
Romero, Walter.		
Sierra Romera, Roberto.		

Calificación: Firma:

Observaciones:

Juaniquina Choque, Rolando.
Ponce, Matias Ezequiel.
Rinaldi, Lautaro Ezequiel.
Romero, Walter.
Sierra Romera, Roberto.

ÍNDICE.

Juaniquina Choque, Rolando.
Ponce, Matias Ezequiel.
Rinaldi, Lautaro Ezequiel.
Romero, Walter.
Sierra Romera, Roberto.

ESTRUCTURA GENERAL DE LA SOLUCIÓN.

Para solucionar el problema propuesto por el enunciado del corrector, decidimos desarrollar seis clases distintas, cuatro son TDA para solucionar el problema en sí mismo (palabra, corrector, diccionario, ListaCandidato), y otras dos son auxiliares para facilitar la implementación de estas (filtro, palabrasinespacio).

El usuario tiene acceso a la clase corrector con sus métodos públicos (que son suficientes para solucionar el problema), encargándose este TDA de relacionar los restantes convenientemente. De este modo, se facilita la tarea del usuario, ya que puede abocarse a lo que realmente necesita sin confundirse con otros tantos métodos de las clases restantes. También accede a palabrasinespacio y a filtro, ya que notamos que no sólo resultan prácticos para implementar las clases, sino que también le sirven al usuario para facilitar la interacción con el corrector.

Describiendo los TDA, podemos decir que un corrector va a estar compuesto por un diccionario y por una lista de candidatos; Y estos a su vez, por palabras. El TDA **corrector** es el responsable de manejar las interacciones entre los diferentes TDA, y con el usuario (ya sea para agregar un libro y ponderarlo, encargarse de que se generen todas las sugerencias válidas para una palabra que haya ingresado el usuario, o para devolverle los primeros cinco candidatos de la lista de sugerencias, ordenados por ponderación descendente).

El TDA **ListaCandidato**, es el que a partir de una palabra dada que escribió el usuario (una vez filtrada y separada convenientemente por espacios), se encarga de encapsularla junto con todos los candidatos posibles a sugerir, generados mediante los métodos de inserción, alteración y borrado, pero como no tiene interacción alguna con el diccionario, no puede encargarse de dejar sólo los válidos (esto sólo puede hacerlo el corrector que tiene acceso tanto al diccionario como a ListaCandidatos).

El TDA **diccionario**, mantiene el orden de las palabras ponderadas, y se encarga de administrar su buen uso, restringiendo las operaciones sobre las mismas (buscarla, saber su ponderación, sumarle uno a su ponderación o agregar una nueva).

El TDA **palabra**, busca encapsular una palabra con su ponderación.

Juaniquina Choque, Rolando.
Ponce, Matias Ezequiel.
Rinaldi, Lautaro Ezequiel.
Romero, Walter.
Sierra Romera, Roberto.

El TDA **filtro** se encarga de extraer de las palabras, ya sean del libro ingresado o las ingresadas por el usuario, los caracteres que no son letras con el fin de permitir el correcto procesamiento de las palabras por el diccionario y por ListaCandidatos (al eliminar un carácter inválido en medio de una palabra, junta los restantes, ya que recibe palabras sin espacios).

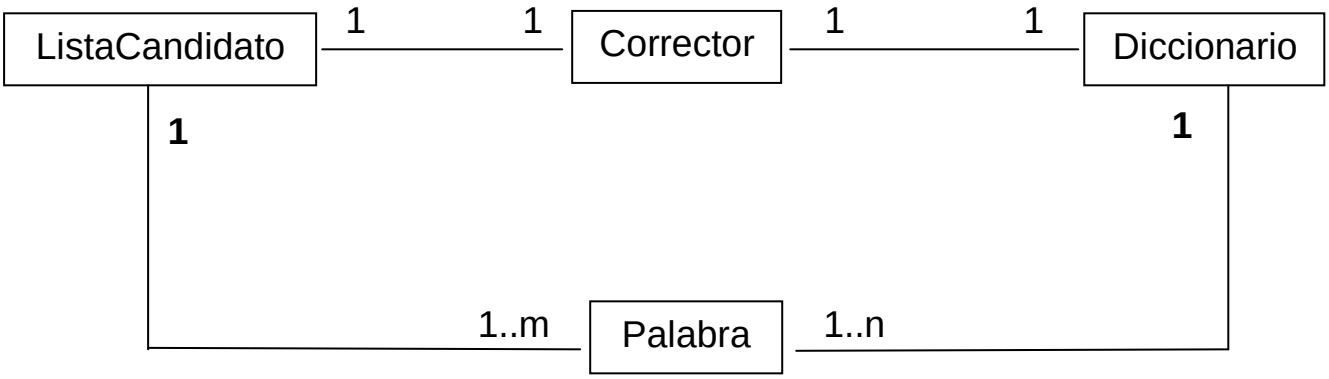
El TDA **palabrasinespacio** tiene como objetivo el procesamiento de los renglones de los libros ingresados, para que las palabras puedan ser filtradas de a una por vez, y también la sucesión de palabras que ingresa el usuario (el corrector corrige de a una palabra por vez), considerando como una palabra a lo que se encuentra entre dos espacios.

Los archivos ‘cambios.rol’ y ‘cambios2.rol’ son utilizados por la clase filtro para reemplazar caracteres en mayúsculas por minúsculas de letras acentuadas, o minúsculas acentuadas por minúsculas sin acento para almacenarlas en el diccionario.

El archivo ‘otros.h’ contiene un enumerado que es utilizado por casi todos los TDA.

Juaniquina Choque, Rolando.
Ponce, Matias Ezequiel.
Rinaldi, Lautaro Ezequiel.
Romero, Walter.
Sierra Romera, Roberto.

DIAGRAMA DE LA SOLUCIÓN.



Queremos aclarar que un objeto de la clase Palabra puede pertenecer a una instancia de la clase ListaCandidato, o a una de la instancia Diccionario (pero no a ambas a la vez); Y por otro lado, la relación 1..1 de Corrector con ListaCandidato es porque decidimos corregir palabra por palabra a medida que las va ingresando el usuario.

Puede notarse la falta de ciertas clases en el diagrama de la solución, esto se debe a que son clases que funcionan de intermediarias o preprocesadoras entre los datos y los TDA.

Juaniquina Choque, Rolando.
Ponce, Matias Ezequiel.
Rinaldi, Lautaro Ezequiel.
Romero, Walter.
Sierra Romera, Roberto.

NOTAS SOBRE EL PROGRAMA.

Para el correcto funcionamiento de la solución al problema propuesto se deberá tener en cuenta lo siguiente:

El archivo libro.txt que se encuentra en la carpeta \datos deberá estar en la misma carpeta que el programa en el momento de la ejecución del mismo (o cualquier otro archivo que desee cargarse para generar un diccionario).

El entorno utilizado para el desarrollo del programa es el CodeBlocks.

El compilador utilizado fue el MinGW.

Los libros a agregarse deben tener extensión "**txt"**, y cuando se le dice el nombre al corrector para que agregue un libro, no hay que escribir la extensión.

Juaniquina Choque, Rolando.
Ponce, Matias Ezequiel.
Rinaldi, Lautaro Ezequiel.
Romero, Walter.
Sierra Romera, Roberto.

CONSIDERACIONES SOBRE LAS ESTRUCTURAS UTILIZADAS.

Cada vez que desea insertarse una palabra en el diccionario, debe previamente analizarse que esta no se encuentre cargada en el mismo (en este caso, se le suma uno a la ponderación), teniendo que recorrer la estructura de datos utilizada para almacenarlas.

En caso de utilizarse una estructura que almacene los datos de manera secuencial, como ser un vector o una lista, deben recorrerse todas las palabras para saber si está (o si estas se encuentran ordenadas, una parte importante de las mismas); En cambio, la estructura de ÁRBOL DE BÚSQUEDA BINARIA (ABB) nos ofrece la ventaja de no tener que recorrerlas todas, ya que guardan una relación de orden entre ellas, pudiendo asegurar que conviene utilizar esta última en el diccionario, porque mejorará los tiempos de búsqueda y de inserción de las palabras.

Nosotros creemos que una ventaja considerable de nuestra implementación del diccionario, es que tenemos un vector para proveer acceso directo según la primera letra de las palabras, teniendo así dividida la carga en 26 (una para cada letra del diccionario), y si suponemos que cada letra tiene aproximadamente la misma cantidad de palabras, estamos acelerando más de 20 veces la búsqueda (ya sea que se implemente con listas o con ABB).

En cambio, en la lista de candidatos necesitamos acceso secuencial, ya que una vez que se generan todos los posibles candidatos para una palabra dada (por alteración, inserción y borrado), y que los mismos no tienen ningún orden asignado, necesitamos controlar uno a uno si se encuentran en el diccionario (en caso contrario deben quitarse de la misma). Como eliminamos la mayor parte de los candidatos generados, quedando una lista reducida de los mismos, y como además el tiempo de eliminación es mucho menor en una lista que en un árbol, creemos que una LISTA es la mejor alternativa de implementación para la lista de candidatos (fundamentalmente, porque no se realizan búsquedas sobre esta, sólo se listan los primeras cinco, que tienen la mayor ponderación).

Como en el segundo trabajo práctico se implementó el diccionario con listas y en el tercero con árboles de búsqueda binaria, y la clase cronómetro se utilizó en ambos, podemos observar que la búsqueda en ABB es mucho más rápida que en listas, multiplicando la velocidad de ejecución del programa varias veces. No hablamos de tiempos porque los mismos varían de una PC a otra.

Juaniquina Choque, Rolando.
Ponce, Matias Ezequiel.
Rinaldi, Lautaro Ezequiel.
Romero, Walter.
Sierra Romera, Roberto.

PRUEBAS UNITARIAS DE LAS CLASES.

El entorno utilizado para el desarrollo de las pruebas es el CodeBlocks.

PRUEBA CORRECTOR

Una vez abierto el programa siga las instrucciones.

Se debe optar por la opción 1 del menú. Cuando le sea pedido ingrese el siguiente texto "libro".

Luego elija la opción 2 para corregir las siguientes palabras (nota: puede utilizar cualquier palabra que desee)

quigote

servantes

(Nótese que las palabras han sido mal escritas de manera deliberada)

Las sugerencias a lo ingresado deben ser:

quijote, quixote, quirote

cervantes

Luego ingrese la opción 3 para salir del programa.

Fin de prueba Corrector.

PRUEBA FILTRO

Ingrese 5 palabras con contengan símbolos no alfabéticos ejemplo: ca{}sa perro? ar?¡?¡bol milane{}-..,sa facultad

Las salidas serán:

casa, perro, arbol, milanesa, facultad

Fin de la prueba Filtro.

Juaniquina Choque, Rolando.
Ponce, Matias Ezequiel.
Rinaldi, Lautaro Ezequiel.
Romero, Walter.
Sierra Romera, Roberto.

PRUEBA LISTA CANDIDATOS

Ingrese una palabra y se listaran todas la opciones generadas por los métodos de inserción, borrado y alteración; Por ejemplo utilice la palabra casa.

Algunas de las salidas serán:

asa

csa

basa

lasa

casar

casam

casaz

Fin de prueba Filtro