Depto. de Computaci'on y Tecnologia de la Informaci'on
CI3391 Laboratorio de Bases de Datos I

MODELACION DE UNA BASE DE DATOS PARA LA RED SOCIAL Soy USB

15 de enero de 2011

Nombre del Profesor :
Ricardo Monascal

Nombre de los Alumnos:
El Assad, Omar Alí 07-40855

De Abreu Molina, Julio 05-38072

Índice

1. INTRODUCCIÓN

Uno de los principales problemas que acontece cuando tenemos un vocabulario, o en su defecto, un diccionario, es cuando necesitamos buscar una palabra, la mayoria de las veces lo hacemos a trav'es de prefijos. Ejemplo: cuando queremos buscar en el diccionario la palabra armamentismo, basta con tomar el prefijo "arma", lo cual nos ahorra la b'usqueda que estamos realizando, porque a partir de ese prefijo, vamos a chequear si la palabra está o no dentro de nuestro vocabulario.
Este problema es atacado por nuestro Consultor Ortografico. El mismo contiene un conjunto de palabras a las cuales llamaremos Vocabulario, y a través de sus diferentes implementaciones (Tries o Arreglos), lograremos resolver algunos problemas, de los cuales se pueden destacar los siguientes: El primero,dado un prefijo, co nsultar todas aquellas palabras que comiencen por dicho prefijo, o que dado un prefijo, te genere el prefijo mas largo.
En el resto del informe lo que podremos ver es lo que correspode al dise'no de ambas variantes, en esta sección podremos encontrar la especificación de ambas implementaciones. Luego, en la siguiente sección, vamos a tener lo que corresponden a los detalles de la implementación, donde vamos a ver peculiaridades con el lenguaje Java. En la siguiente sección haremos un análisis del estado actual de nuestro proyecto, donde veremos si existen anomalías, y cuáles son. Finalmente tendremos las conclusiones y las Fuentes Bibliográficas.

2. DISENO

El siguiente codigo es la especificación del ConsultOrt (Interfaz del Consultor):

public interface ConsultOrt

public normal behavior
requires
bf(p) y vocabulario.has(new JMLString(p));
ensures
vocabulario.equals
(
old(vocabulario).union
(
new JMLValueSet(new JMLString(p))
)
);
assignable
vocabulario;
also public exceptional behavior
requires
!bf(p) —— vocabulario.has(new JMLString(p));
signals only
ExcepcionPalabraNoBienFormada,
ExcepcionPalabraYaRegistrada;
signals
(ExcepcionPalabraNoBienFormada) !bf(p);
signals
(ExcepcionPalabraYaRegistrada)
vocabulario.has(new JMLString(p));
assignable
nothing;
/
public void agregar(String p) throws
ExcepcionPalabraNoBienFormada,
ExcepcionPalabraYaRegistrada;

public normal behavior
requires
bf(pr);
ensures
vocabulario.equals(old(vocabulario))
y
(forall int i, j ; 0 ¡= i ¡result.length y 0 ¡= j ¡result.length y i != j ; !result[i].equals(result[j]))

```
y (forall int i ; 0 ¡= i ¡result.length ; vocabulario.has ( new JMLString(result[i])
) )
y (forall int i ; 0 ¡= i ¡result.length ; ipf(pr, result[i]) );
assignable
nothing;
also public exceptional behavior
requires !bf(pr);
signals only
ExcepcionPalabraNoBienFormada;
signals
(ExcepcionPalabraNoBienFormada) !bf(pr);
assignable
nothing;
/
public String[] consultarPorPrefijo(String pr) throws
ExcepcionPalabraNoBienFormada;

    public normal behavior
requires bf(pl);
ensures vocabulario.equals(old(vocabulario)) y ipf(result, pl) y (forall String s ;
vocabulario.has(new JMLString(s)) ; ipf(s, pl) ==¿ipf(s, result) );
assignable nothing;
also public exceptional behavior
requires !bf(pl);
signals only
ExcepcionPalabraNoBienFormada;
signals
(ExcepcionPalabraNoBienFormada) !bf(pl);
assignable
nothing;
/
public String prefijoMasLargo(String pl) throws
ExcepcionPalabraNoBienFormada;

    /* requires true;
ensures result ¡==¿(p !=  y (forall int i ; 0 ¡= i ¡p.length() ; 0 ¡= p.charAt(i) -
'a' y p.charAt(i) - 'a' ¡= 25 )); */
public /*@ pure @*/ boolean bf(String p);

    /* requires bf(p) y bf(q);
ensures result ¡==¿p.length() ¡= q.length() y (forall int i ; 0 ¡= i ¡p.length() ;
p.charAt(i) - q.charAt(i) == 0 ); */
public /*@ pure @*/ boolean ipf(String p, String q);

    public void leerArchivo (String s) throws NoTextFoundException;
```

public void listar(ConsultOrt a);

public Iterator iterator();

Lo que sigue es la especificacion del tad Cola, la cual la usamos para Consultar por prefijos en la Variante Trie:

import org.jmlspecs.models.JMLValueSequence; public interface ColaInterface

assignable nothing;

public void encolar (String a);

public int tamano();

public String[] formQeueToArray();

El TAD Cola lo estamos usando ya que para consultar por prefijo, llamamos recursivamente a una función la cual llamamos toQeue, la cual toma las palabras que comienzan por un prefijo dado, y los mete en una cola, esto es para luego llevarlos a un arreglo. Cabe destacar que para el TAD Cola, usamos el tipo concreto Lista, esto es para evitar el tamano limitado de los arreglos.

3. DETALLES DE IMPLEMENTACION

Realmente no hay pecularidades que resaltar del lenguaje Java, salvo algunos detalles como lo son los Warnings que arroja JML, esto se debe a que hay especificaciones informales que JML obviamente no reconoce. Adicionalmente, dada la costumbre de que en Java teniamos métodos, gracias a los cuales, dentro de un tad acostumbramos a poner this.¡Nombre del Procedimiento¿. En ConsultOrtTriesArreglos, hemos podido encontrar algunos procedimientos, en los cuales se les pasa como parámetro una variable del tipo abstracto que estamos implementando. Esto lo que hace es que a la hora de llamarlos, ya no podemos llamarlos como llamamos a los procedimientos tradicionalmente, sino que se le pasa el parámetro del tipo.

4. ESTADO ACTUAL

Luego de hacer todas las verificaciones posibles, podemos decir que nuestro Consultor funciona perfectamente , esto significa que no presenta ninguna anomaías en ninguna de las dos variantes. Se comprobó cada una de las operaciones, tanto para tries como para arreglos, y funcionan con total normalidad.

Ahora veamos como funciona nuestro Consultor.

MANUAL DE USO DE CONSULTOR ORTOGRAFICO

En primer lugar, se debe contar con los siguientes archivos en una carpeta: Console.java, ConsultOrt.java, ConsultOrtArreglos.java, ConsultOrtTriesArreglos.java y Cliente.java. Luego, abrir una consola y situarse en la carpeta correspondiente.

Luego, a través del siguiente comando, se procederá a compilar cada uno de los archivos: jmlc *.java. Con esto, se van a generar los archivos .class, sin los cuales no se podrá ejecutar el programa principal.

El paso siguiente es poner en la Consola el comando jmlrac Cliente. Esto nos llevará al menú de opciones. La primera opcion nos ofrece crear un consultor a traves de dos variantes: TRIES O ARREGLOS. Este es el paso mas importante, ya que al principio el Consultor no se encuentra inicializado, y si ejecutamos cualquiera de las otras opciones, nos va arrojar una excepción de java muy conocida: NULL POINTER EXCEPTION.

Las otras opciones que tenemos en el menu disponibles son las siguientes: Agregar manualmente las palabras al vocabulario, en esta opción, le damos la opción de poner la cantidad de palabras a ingresar al vocabulario del consultor, y luego por teclado tiene que ir poniendo cada una de las palabras. Cabe destacar que si coloca palabras que contengan caracteres especiales, algunas o todas las

letras en mayúscula, o le das a espacio y enter dando a entender al consultor que vas a insertar una palabra vacia, va a arrojar una excepción, porque nuestro consultor sólo admite palabras en minúsculas, no vacías y sin caracteres especiales. Adicionalmente, si insertas una palabra que ya estaba agregada, también arroja una excepción, ya que nuestro consultor no admite palabras repetidas.

La siguiente opción es Consultar por Prefijo. En esta opción le debes pasar una palabra, nuevamente en minúscula, noacía y sin caracteres especiales, y nuestro consultor procederá a buscar todas aquellas palabras que comiencen por la palabra ingresada. En caso exitoso, lo que ocurrirá es que se imprimirán en pantalla todas las palabras encontradas. En caso contrario, le diremos que no hay coincidencias.

Nuestra próxima opción es Prefijo más Largo. Aquí debe ingresar una palabra en minúsculas, no vacía y sin caracteres, y nuestro consultor procederá a chequear si hay alguna palabra que comienza por dicha palabra. En caso exitoso, se procederá a imprimir en pantalla la palabra ingresada. En caso contrario, se corta la palabra una letra de derecha a izquierda, para realizar el mismo procedimiento antes descrito. Esto lo va a hacer hasta que llegue al principio de la palabra.

La siguiente opción es Cargar un archivo. Aqui le pediremos que ingrese la dirección COMPLETA de la ubicación del archivo a cargar. Si el archivo no existe, o la dirección es incorrecta, el programa arroja una excepción ya que el archivo no fue encontrado o no existe.

La próxima opción es listar el vocabulario que tenemos hasta ahora. Si le das a esta opción, se imprimiran todas las palabras en pantalla.
La última opción es Salir del Programa.

5. CONCLUSIONES

A lo largo del proyecto hemos podido aprender acerca del lenguaje JAVA y algunos de sus comandos. Hemos podido aprender sobre el uso de clases, el manejo de excepciones en JAVA, a decir verdad fue interesante trabajar paralelamente con arreglos y arboles Tries, la complejidad que tiene recorrer arboles con nodos de arreglos de 26 posiciones, la funcionabilidad de recorrer el arbol ordenadamente y así obtener las palabras listadas en orden lexicográfico, no se puede obviar el hecho de trabajar bajo JML que presenta una dificultad extra al agregar al lenguaje la posibilidad de crear **invariantes** y **funciones de cota** que permiten la verificacion de los codigos y lo que modifican, aparte de verificar la terminación de ciclos.

A lo largo del proyecto nos hemos encontrado varias dificultades, ya que el manejo de referencias en JAVA requirió investigación para su buena implementación. Para la implementación con Tries, los iteradores sobre el árbol tambien formaron parte de las dificultades.

Referencias

LISKOV, Barbara: "Program Development in Java - Abstraction, Specification, and Object-Oriented Design", Addison-Wesley, 2001.

API JAVA 1.4 http://java.sun.com/j2se/1.4.2/docs/api/