UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA

LENGUAJE Y MÉTODOS DE PROGRAMACIÓN

SISTEMA DE RECUPERACION DE INFORMACION EN PROGRAMACION FUNCIONAL

Profesor: Roberto González
Alumno: Roberto Orellana

Fecha de Entrega: 22/10/2017

Santiago de Chile

21 - 10 – 2017

TABLA DE CONTENIDOS

CAPÍTULO 1. INTRODUCCION

El siguiente informe trata de explicar cómo llegar a la solución del laboratorio N°2 de Lenguaje y métodos de Programación.

1.1 ANTECEDENTES Y MOTIVACIÓN

El laboratorio consta de crear un Buscador en cuatro lenguajes distintos de programación, siendo este el segundo en Lenguaje de Programación Funcional (Scheme) en el entorno de desarrollo Dr.Racket., lo cual me agrada porque ahora me es muy familiar.

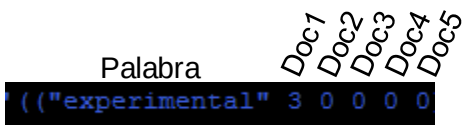
1.2 DESCRIPCIÓN DEL PROBLEMA

El Problema consiste en crear un sistema de recuperación de Información en el Lenguaje de Programación Funcional (Scheme), el cual lo dividimos en sub-problemas como crear un índice, consultar una palabra, consultar una frase, ordenar los resultados de manera ascendente y descendente según la cantidad de palabras encontrados en cada documento, e imprimir el resultado de la forma más legible posible para el usuario.

1.3 SOLUCIÓN PROPUESTA

Para obtener una solución real al problema que se propone, se estudió cuidadosamente los requerimientos funcionales y no funcionales contenidos en el enunciado del Laboratorio, y de esta forma llegar a una posible solución comenzando por la implementación apropiada para los problemas a través del TDA, luego la creación del Índice que es la base de todas las funciones, se utiliza como representación lista de listas que contienen elementos (índice disperso), e ir acoplando todas las otras funcionalidades que contiene el buscador debido al no uso de variables donde pueda almacenar los resultados.

Crear Índice.
Para la creación del índice la representación elegida fue una lista de lista (matriz dispersa) de elementos, en el cual el primer elemento corresponde a la palabra y el resto de la lista corresponde a la cantidad de veces que se repite la palabra en cada documento.



Consultar Palabra y Frases
Para la consulta de palabras es muy simple solo debes comparar la palabra con las del índice y te mostrara los documentos que contiene cada palabra, la de la frase es muy parecida a la de la creación del índice donde debes separar cada palabra de la frase, filtrar las palabras con las stopwords y sumar la cantidad de palabras en cada documento para mostrar el resultado.

Ordenar Ascendente o Descendente.
Con el resultado de la búsqueda, ordena los documentos a través de la cantidad de palabras encontrada en cada documento ascendente o descendentemente como prefiera el usuario.

Mostrar Resultados.
Muestra por pantalla los resultados de la manera más legible posible para el Usuario

1.4 OBJETIVOS Y ALCANCES DEL PROYECTO

1.4.1 Objetivo General

El Objetivo general del proyecto consta en lograr el correcto funcionamiento del programa a través del lenguaje de programación Scheme, utilizando lo aprendido en clases y laboratorio sobre paradigma de programación Funcional.
Conocer el correcto uso de cada código utilizado en cada función, reconocer el tipo de recursión, funciones anónimas como lambda, encapsulación de la función y reconocer las capas de la estructura del TDA (Constructores, Funciones de Pertenencia, Selectores, Modificadores y Otras Funciones).

1.4.2 Objetivos Específicos

Lograr que el usuario (con conocimientos básico en programación de Scheme (DrRacket)) pueda crear un índice limpio libre de Stopwords y caracteres que compliquen la búsqueda, hacer una consulta de una frase compuesta o palabra, y mostrar el resultado de la búsqueda en un ranking de forma ascendente o descendente según la cantidad de palabras encontradas en cada documento y Mostrar por pantalla los resultados de la manera más legible posible para el Usuario

1.4.3 Alcances

Obtener un correcto funcionamiento del programa gracias a los objetivos y problemas que se plantearon anteriormente.

1.5 METODOLOGÍA Y HERRAMIENTAS UTILIZADAS

Las herramientas y técnicas utilizadas para la solución de final del problema constan de lo aprendido en clases (TDA, recursiones, funciones anónimas, encapsulación, etc), investigación propia del alumno y las metodologías enseñadas por el profesor, analizando el problema y teniendo una abstracción de cómo resolverlo.
El Programa utilizado en mi caso fue la de DrRacket que gracias a su fácil compilación y ejecución hace que sea una interfaz más amigable para el programador.

CAPITULO 2. FUNDAMENTOS TEORICOS

2.1 definiciones

- **Paradigma de programación Funcional:** La programación Funcional es un paradigma basado en el uso de funciones matemáticas, la programación funcional tiene sus raíces en el cálculo lambda, es un lenguaje donde las variables no tienen estado, no hay cambios en éstas a lo largo del tiempo y son inmutables. Además los programas se estructuran componiendo expresiones que se evalúan como funciones.

- **Función:** una función es un grupo de instrucciones con un objetivo en particular y que se ejecuta al ser llamada desde otra función o procedimiento. Una función puede llamarse múltiples veces e incluso llamarse a sí misma (función recurrente).

- **Funciones de orden superior:** Funciones de orden superior son funciones que pueden tomar otras funciones como argumentos o devolverlos como resultados.

- **Recursividad:** Las funciones recursivas se invocan a sí mismas, permitiendo que una operación se realice una y otra vez hasta alcanzar el caso base.

- **StopWords:** (palabras comunes) son términos que los buscadores ignoran cuando rastrean un texto.

- **Índice:** el Índice me permite encontrar información en documentos y colecciones de documentos.

- **Funciones Anónimas:** la funciones anónimas (o subrutina) constituyen una forma muy flexible de crear funciones sobre la marcha

2.2 Métodos Utilizados

El Método Utilizado para representar o modelar la creación del programa fue el **TDA** (tipo de dato Abstracto)

- **Representación**: el Índice disperso es representada por una lista de lista de elementos, donde el primer elemento de la lista corresponde a la palabra del índice y el resto de la lista cada elemento corresponde a la cantidad de veces que se repite la palabra en cada documento.

- **Constructores:** utilice funciones que me permitieron construir la lista de lista de elementos donde cada palabra tiene la cantidad de elementos igual a la cantidad de documentos

- **Funciones de Pertenencia:** funciones que me indican si el elemento es una lista y me comprueba si dos palabras son iguales

- **Selectores:** función que me permite seleccionar el primer elemento de la lista para ver si es igual a la palabra consultada

- **Modificadores:** funciones que me permite modificar un string con muchas frases en varios string de palabras.

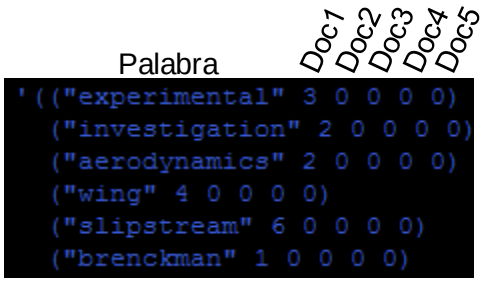
- **Otras Funciones:** funciones que me permiten mostrar el documento por pantalla y ordenar de la forma ascendente o descendente.

CAPITULO 3, DESARROLLO DE LA SOLUCION

3.1 Análisis

En el análisis la deducción principal es la separación del texto en palabras (string) para la creación del Índice Invertido, que gracias a estos se podrá trabajar fácilmente sobre las otras funciones del programa.

Representación: el Índice disperso es representada por una lista de lista de elementos, donde el primer elemento de la lista corresponde a la palabra del índice y el resto de la lista cada elemento corresponde a la cantidad de veces que se repite la palabra en cada documento.



Constructores: utilice funciones que me permitieron separar las palabras, construir una lista con todas las palabras, eliminar las que se repiten y asi dejar solo una de ellas, luego se comprueba cuantas veces se repite la palabra en cada documento, de esa manera se construye la lista de lista de elementos donde cada palabra tiene la cantidad de elementos igual a la cantidad de documentos

Funciones de Pertenencia: funciones que me indican si el elemento es una lista y me comprueba si dos palabras son iguales para eliminar en el caso de crear el índice las palabras que se repiten o para el caso de la stopwords, para eliminar las palabras comunes.

Selectores: función que me permite seleccionar el primer elemento de la lista para ver si es igual a la palabra consultada a través del car de cada lista se logra sacar el elemento que corresponde a la palabra

Modificadores: funciones que me permite modificar un string con muchas frases en varios string de palabras y modifica el resultado de los documento al sumar a través de la función anónima **(map (lambda (L1 L2) (+ L1 L2)) (cdr (car index)) result)** el resultado del índice de dos o más palabras que se estén consultando.

Otras Funciones: funciones que me permiten mostrar el documento por pantalla de forma legible para el usuario y ordenar de la forma ascendente o descendente gracias a la función anónima **(sort results (lambda (X Y)(> (car X) (car Y))))**

3.2 Diseño

Podríamos decir que el programa no tiene diseño o interfaz gráfica ya que funciona en base a llamadas de funciones, a través de interfaz de DRracke se puede hacer el llamado a las distintas funcionalidades del programa, CreateIndex, TermQuery, PhraseQuery, Ranking y Results -> string, al ingresar cada una de estas funciones se debe ingresar con los parámetros básicos mencionados en el siguiente ejemplo y tener conocimiento básico sobre programación de Scheme, por ejemplo:

CreateIndex

Esta función crea un índice, el Índice me permite encontrar información en documentos y colecciones de documentos, esta función solicita 2 argumentos como parámetros de entrada, documents y stopwords que son una lista de documentos y palabras que vienen precargados en el programa. Por ejemplo:

documents

```
(define doc '('(1 "titulo1" "autores1" "biblio1" "texto numero uno")
              '(2 "titulo2" "autores2" "biblio2" "texto numero dos")
              '(3 "titulo3" "autores3" "biblio3" "texto numero tres")
              '(4 "titulo4" "autores4" "biblio4" "texto numero cuatro"))
```

Stopwords

```
(define ListaStopWords '("a" "about" "above" "across" "after" "afterwards" "again" ....
```

La función se llamaría de la siguiente manera:

```
(createIndex ListaDocumentos ListaStopWords)
```

TermQuery

Esta función solicita 3 argumentos como parámetros de entrada, index que corresponde al índice entregado por la función createIndex, term corresponde al string que se consulta, y documents que sirve para retornar la lista de documentos que contengas incidencias con la palabra consultada. Por ejemplo:

index: corresponde al índice disperso (createIndex ListaDocumentos ListaStopwords)
term: corresponde a la palabra a consultar. (“wing”)
documents: corresponde a toda la lista de documento precargados. (ListaDocumentos)

```
(termQuery index term documents)
```

Se llamaría la función de la siguiente manera:

```
(termQuery (createIndex ListaDocumentos ListaStopWords) "wing" ListaDocumentos)
```

PhraseQuery

Esta función solicita 4 argumentos como parámetros de entrada, index que corresponde al índice entregado por la función createIndex, stopwords corresponde a la lista de palabras comunes que se eliminan en la consulta, phrase corresponde al string(frase) que se consulta, y documents que sirve para retornar la lista de documentos que contengas incidencias con la palabra consultada.
En este ejemplo usaremos otra lista de documentos (doc) para que el ejemplo sea mejor visible
Por ejemplo:

index: corresponde al índice disperso (createIndex doc ListaStopwords)
stopwords: corresponde a las palabras comunes (ListaStopwords)
phrase: corresponde a las palabra o frase a consultar. ("cuatro titulo2 of dos texto uno")
documents: corresponde a toda la lista de documento precargados. (doc)

```
(phraseQuery index stopwords phrase documents)
```

Se llamaría la función de la siguiente manera:

```
(phraseQuery (createIndex doc ListaStopWords) ListaStopWords "cuatro titulo2 of dos texto uno" doc)
```

Ranking

Esta función solicita 2 argumentos como parámetros de entrada, results que corresponde al resultado entregado por la función TermQuery o PhraseQuery, orderType que corresponde al orden ascendente o descendente.
En este ejemplo usaremos otra lista de documentos (doc) para que el ejemplo sea mejor visible
Por ejemplo:

result: corresponde al resultado de la consulta (termQuery o phraseQuery)
orderType: corresponde al orden ascendente(2) o descendente (1)

```
(ranking results orderType)
```

Se llamaría la función de la siguiente manera:

```
(ranking (phraseQuery (createIndex doc ListaStopWords) ListaStopWords "cuatro titulo2 of dos texto uno" doc) 1)
```

Results->String

Esta función solicita 1 argumento como parámetro de entrada, results que corresponde al resultado entregado por la función Ranking TermQuery o Ranking PhraseQuery.
En este ejemplo usaremos otra lista de documentos (doc) para que el ejemplo sea mejor visible
Por ejemplo:

result: corresponde al resultado de la consulta (Ranking(termQuery) o (Ranking(phraseQuery)

```
(Results->String results)
```

Se llamaría la función de la siguiente manera:

```
(Results->String (ranking (phraseQuery (createIndex doc ListaStopWords) ListaStopWords "cuatro titulo2 of dos texto uno" doc) 1))
```

CAPITULO 4. RESULTADOS

Se da cumplimiento a los objetivos del proyecto, creando las funciones requeridas; crear un índice a partir de los documentos y StopWords precargados en el programa, limpiar el archivo eliminando las palabras comunes, realizar consultas de frases o palabra y eliminar las palabras comunes de la consulta, ordenar de manera ascendente o descendente dependiendo del parámetro ingresado por el usuario e imprimir los resultados de la manera más legible posible para el usuario.

CAPITULO 5. CONCLUSION

Gracias a este laboratorio logre adquirir más conocimientos sobre este paradigma y lenguaje funcional en cuanto funciones, recursividad natural (con estados pendientes) o de cola (sin estados pendientes), el caso base, proceso recursivo y condición de borde, funciones anónimas y funciones de orden superior (encapsulamiento), además de trabajar sin estados o variables y La forma en que las subrutinas se pueden volver a utilizar desde distintas funciones.
El uso del TDA se hace fundamental a la hora de representar o modelar la creación de un programa, ya sea en programación funcional como el que acabamos de hace, o cualquier otro lenguaje y método de programación.

CAPITULO 6. REFERENCIAS

Web: https://docs.racket-lang.org/reference/

Web: http://www.udesantiagovirtual.cl/moodle2/course/view.php?id=9365

Web: https://www.youtube.com/watch?v=NwZPlLTK-UQ&list=PLTd5ehIj0goMswKV4Glhr4uixrhCmihIt

Libro: Programando con Racket 5 de Eduardo Navas, Departamento de Electrónica e Informática, Universidad Centroamericana.

Libro: Larson, Jim, *An Introduction to Lambda Calculus and Scheme.*