|  |
| --- |
| **PARADIGMAS DE PROGRAMACIÓN**  **LABORATORIO 2**  **PARADIGMA FUNCIONAL** |
|  |
| **Realizado por: Jorge Luis Plaza Contreras** |

Profesor: Roberto González

|  |  |  |
| --- | --- | --- |
|  | Santiago de Chile |  |
|  | 2 - 2017 |  |

Tabla de Contenidos

[Tabla de Contenidos I](#_Toc495957433)

[Índice de Figuras I](#_Toc495957434)

[CAPÍTULO 1. Introducción 1](#_Toc495957435)

[1.1 Paradigma funcional 1](#_Toc495957436)

[1.2 Descripcion del problema 1](#_Toc495957437)

[CAPÍTULO 2. Descripción de la solución 3](#_Toc495957438)

[2.1 Analisis del problema 3](#_Toc495957439)

[2.2 Diseño de la solucion 4](#_Toc495957440)

[2.2.1 Implementación TDA 4](#_Toc495957441)

[2.2.3 Constructor 5](#_Toc495957442)

[2.2.4 termQuery 6](#_Toc495957443)

[2.2.5 phraseQuery 6](#_Toc495957444)

[2.2.6 Ranking 7](#_Toc495957445)

[2.2.7 Results->String 7](#_Toc495957446)

[2.2.8 Otras funciones 8](#_Toc495957447)

[2.3 Aspectos de implementación 8](#_Toc495957448)

[CAPÍTULO 3. Análisis de resultados 9](#_Toc495957449)

[CAPÍTULO 4. Conclusión 11](#_Toc495957451)

[CAPÍTULO 5. Referencias 12](#_Toc495957452)

Índice de Figuras

[Figura 2‑1: Ejemplo indice 4](#_Toc382443674)

[Figura 2‑2: Seudocodigo ordenamiento burbuja 7](#_Toc382443674)

# Introducción

Durante el semestre se trabajar con distintos paradigmas de programación, de los cuales, el objetivo es implementar cuatros de estos paradigmas en distintos lenguajes de programación. El programa a implementar es un sistema de recuperación de información.

Paradigma funcional

El paradigma funcional se define como un conjunto de definiciones de funciones matemáticas. No existe el concepto de variable, por lo tanto, no existe un lugar en la memoria que se pueda modificar, no es posible hacer iteraciones como en el paradigma imperativo.

Como característica principal, la evaluación de las funciones se hace a través de la recursión y condiciones. Se respeta el dominio y recorrido de las funciones. Uno de los típicos elementos es la composición de funciones.

Para implementar el programa se utiliza el lenguaje de programación Scheme mediante DrRacket.

Descripcion del problema

El programa, en términos generales, define dos funciones de entrada. En uno de ellos vienen distintos documentos con su título, autores y un breve texto. Cada uno de estos documentos tiene que ser archivado en un índice para posteriormente, mediante una consulta del usuario, buscar la mayor cantidad de repeticiones de la consulta dentro de cada documento. Finalmente, los resultados tienen que ser ordenados en un ranking los resultados al usuario.

La segunda función contiene una lista con StopWords, es decir, palabras que no se deben considerar al momento de crear el índice o realizar la consulta.

Se toman en cuenta dos tipos distintos de consulta, uno de ellos se realiza mediante solo una palabra, y la otra consulta, mediante una frase que el usuario ingresara como parámetros a las funciones definidas en el programa. Se debe dar la opción de un TDA para el índice y una forma de mostrar los resultados de forma más ordenada para la vista del usuario.

# Descripción de la solución

2.1 Analisis del problema

El problema está constituido por distintas partes. Se distingue entre los requerimientos No funcionales y funcionales.

Con respecto a los requerimientos no funcionales, se debe dar énfasis al comentado de las funciones especificando de manera correcta el tipo de recursión a utilizar. En este caso, se debe realizar una especificación de cada función con su dominio y recorrido. En cada llamado recursivo se especifica su tipo de recursión (línea, de cola o arbórea).

Para cada función de debe respetar su dominio y recorrido como también aplicar dos tipos de recursividad en las funciones y aplicar funciones anónimas. Al final de código se da ejemplos de todas las funciones aplicadas en el programa.

Se tiene que facilitar un archivo de texto con la autoevaluación de cada una de las funciones y respetar la estructura de las funciones señaladas en el enunciado del proyecto.

Ya hablando de los requerimientos obligatorios funcionales. Se requiere implementar 6 elementos importantes, de los cuales se tiene:

* Implementación TDA: Se necesita una buena representación de índice para lograr una mayor efectividad en el código, esto implica crear funciones de pertenencia, selectoras, modificadoras que cumplan con el requisito establecido.
* createIndex: Es parte del TDA y se encarga de construir un índice mediante la lista de documentos y stopwords. Se tiene que asegurar que la representación sea acorde con la creación del índice.
* termQuery: Función que recibe un índice y una consulta de una palabra ingresada como string y se encarga de devolver una lista con los resultados medidos mediante un número (RankValue). Para este caso se necesita recorrer cada parte del índice encontrando la palabra ingresada por el usuario y aumentar el valor de este para cada documento por separado.
* phraseQuery: Función que recibe un índice, la lista de stopwords y una frase del usuario como string. Se requiere separar la frase por palabra, para luego recorrer cada palabra de la consulta encontrando coincidencias en el índice aumentando el valor del RankValue.
* Ranking: Función que recibe un resultado y un número. Ordena los resultados de forma ascendente (1) o descendente (2) dependiendo del parámetro recibido. Se requiere identificar el número ingresado, validarlo, y usar un algoritmo de búsqueda que permita ordenar los resultados de forma adecuada.
* Results->String: Función que recibe una lista de resultados y se encarga de mostrarlos de forma legible al usuario. Se requiere agregar saltos de línea en cada espacio del índice, para luego unir todo y mostrarlos al usuario.

2.2 Diseño de la solucion

Se aborda mediante el método de resolución llamado división en sub-problema. Cada requerimiento del problema se toma como un pequeño sub-problema y algunos de estos sub-problema se dividen en más problemas. Esto ayuda a encontrar una solución global al sistema de recuperación de información.

### 2.2.1 Implementación TDA

Se aborda el primer problema presentado, crear un TDA para la implementación de un índice invertido. El primer sub-problema a abordar es la representación de dicho índice. Tomando en cuenta las condiciones pedidas para crear respectivo índice, se representa mediante una lista de listas. Donde cada sub-lista contiene 5 elementos. El primer elemento es la ID del documento. El segundo es una lista con las palabras del título del documento, el tercero una lista con las palabras del autor, el cuarto con las palabras de la bibliografía y el quinto elemento, una lista con las palabras del texto del documento. Cada una de las listas no tendrá incluidos las palabras consideradas como stopwords. Cabe destacar que esta representación se decidió antes de la modificación del enunciado.

*Figura 2-1: Ejemplo indice*

Ya el segundo sub-problema es crear una función de pertenencia. Se define como index?, su entrada es una índice y devuelve un bool. Se validara si cada lista de la sub-lista cumple con la condición de índice. Se realiza mediante una recursión de cola, entregando en cada llamado una lista para validar. Si alguna lista no cumple, entonces devuelve falso, en caso contrario, devuelve verdadero.

Otro problema a abordar es la creación de selectores. Se crean selectores para cada parte del documento (título, autor, bibliografía y texto). Para crear selector para título, se pide como entrada el índice y un número que representa el documento del cual se quiere obtener. El problema se aborda mediante recursión de cola. El caso base es cuando el número es igual a cero. Aquí se devuelve el segundo elemento de la cabeza del índice, caso contrario, se llama recursivamente pasando la cola del índice y reduciendo el número en 1. Ahora, aplicando el método de resolución de problemas similitud débil, se procede a realizar el resto de selectores solo cambiando el valor de entrega en la función.

### 2.2.3 Constructor

Como se mencionó, createIndex es un requerimiento funcional obligatorio que forma parte de la implementación del TDA de la estructura del índice invertido. Recibe la lista de documentos y de stopwords y devuelve el índice mencionado anteriormente. Se define una función auxiliar (createIndexAux) dentro de ella que se encarga de crear una lista con el ID de documento y la lista de palabras. Esta función se apoya de otra que se encargara de eliminar los stopwords de la lista (función). Para el caso de createIndexAux se utiliza dos llamado recursivos lineales, que se diferencian solo cuando la lista contiene el ID y cuando contiene el string correspondiente a una parte especifica del documento. Fuera de la función auxiliar se realiza otro llamado recursivo lineal que une todas las listas para formar el índice. Cada llamado recursivo se considera lineal por contener el estado pendiente cons, que corresponde a un recurso de Racket.

### 2.2.4 termQuery

Para abordar este problema, se utiliza nuevamente el método de resolución división en sub problemas. Gracias al análisis del problema se detectan dos sub problemas para abordar. El primero de ellos es el cálculo del número RankValue y el segundo es agregar este valor a la cabeza de cada sub lista del índice invertido creado. Para el primer problema se usara el siguiente criterio, si la palabra consultada se encuentra en el título, autor, bibliografía o texto se sumara 1 al valor sin tomar en cuenta cuantas veces se repita la palabra en cada una de ellas. Esto lo hace una sub función anónima dentro de termQuery, mediante llamados recursivos de cola y con el apoyo de una función llamada estaEnLista que valida si un string se encuentra dentro de una lista se crea el RankValue.

Luego de obtener RankValue, se requiere agregar este valor a la cabeza de la lista en el índice. Esto lo realiza la función anónima listaUnida. Mediante llamado recursivos lineales se va formando la lista junto con el RankValue.

Cabe destacar que los resultados entregados y el llamado a ambas funciones explicadas anteriormente lo hace la funcion termQuery en las últimas líneas de la función. Mediante un llamado recursivo lineal va llamando recursivamente a ambas funciones y así misma para entregar la lista según lo pide el enunciado.

### 2.2.5 phraseQuery

phraseQuery se aborda mediante división en sub problemas y similitud débil por su parecido a la función termQuery. Esta vez se requiere agregar otro sub problema, este es hacer un desglose de la frase obtenida por el usuario creando una lista con palabras que no contenga stopwords en ella. Para esto se crea una función anónima llamada “frase”, esta función, mediante la ayuda de la función esStopWord, transforma la consulta en una lista de String sin los stopword en ella.

Luego, los siguientes sub problemas son similares a los mencionados en termQuery, con pequeñas diferencias. El criterio de RankValue es parecido al anterior, solo que en este caso si algunas de las palabras de la lista se encuentra en algunas de las partes del documentos, entonces se suma 1, no toma en cuenta cuantas veces se repite la palabra. Entonces las funciones número y listaUnida, son creadas mediante similitud débil con respecto a las funciones creadas en termQuery. Generando llamados recursivos lineales como de cola.

Finalmente, phraseQuery realiza los llamados correspondiente para crear la lista de resultados de la forma en la que aparece en el enunciado.

### 2.2.6 Ranking

Para realizar la función ranking, se debe implementar un algoritmo de búsqueda. El algoritmo escogido para este caso es el ordenamiento burbuja. Este procedimiento se realiza mediante la comparación de los elementos consecutivos dentro de la lista, moviendo el elemento correspondiente a su posición tomando en cuenta el orden requerido por el usuario. Se hace la distinción entre ambos en la función mediante los números 1 y 2 como se pide en el enunciado. El algoritmo de implementa mediante llamados recursivos lineales, dejando estados pendientes “cons”.

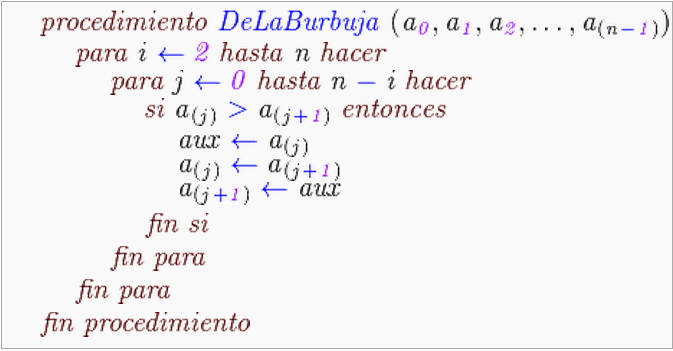


Figura 2-2: Seudocódigo del ordenamiento burbuja

### 2.2.7 Results->String

Para la implementación de esta función se hace uso de funciones como string-join, string-append y display. Que se encargaran de unir, agregar y mostrar posteriormente al usuario.

Se separa en dos sub problemas. El primero es unir cada parte de la lista del índice como un string. Esto requiere transformar toda la lista a string que se hace mediante una función llamada lista->string, luego se agrega el salto de línea correspondiente al final de la lista. El segundo problema es agregar cada documento para formar un string completo, esto se hace mediante la función “s”, que mediante una recursión lineal va formando el string para que posteriormente la funcion Results->String lo muestre al usuario mediante display.

### 2.2.8 Otras funciones

Distintas funciones de usan para apoyar la implementación de los requerimientos obligatorios del laboratorio, entre ellos se menciona: esStopWord, que recibe la lista de stopword y un string y retorna un bool que verifica si la palabra ingresada se encuentra en la lista. Función que recibe una lista de palabras y la lista de stopword y retorna la lista sin stopwords. estaEnLista es una función que recibe una lista de string, un string y valida si el string se encuentra en la lista mediante un bool. Finalmente se encuentra lista->string que recibe una lista y transforma todos los numero que están en ella a string.

## 2.3 Aspectos de implementación

El programa tiene 14 funciones en total, cada una de ellas realizadas aplicando reglas del paradigma funcional y recursos de Racket. Entre estos recursos se menciona aplicación de distintas funciones tales como: string-join, une una lista de string en un string completo. String-append, función que agrega un string a otro. Cons, se encarga de crear una lista a través de dos parámetros, se utiliza mucho en las recursiones aplicadas en el programa. Car y cdr también se usan demasiado ya que sirven para obtener los elementos de las listas implementadas. String-Split que separa un string en una lista de palabras, entre otras.

Todas estas funciones de tomaron en cuenta debido, principalmente, a la elección usada para la representación del índice, como los parámetros de entrada que requería el enunciado del proyecto, tales como, la lista de documentos y la lista de stopwords.

Todas las secciones del código se encuentran comentadas, especificando entrada y salida de funciones como llamados recursivos y sus tipos de recursividad. Cada uno tiene la explicación del porque es una recursión lineal o de cola, que fueron las recursiones utilizadas en el proyecto.

# Análisis de resultados

Los requerimientos obligatorios del proyecto son los siguientes:

1. Implementación en lenguaje de programación Scheme (Racket): Cumple totalmente con el requisito.
2. No hacer uso de set! o funciones similares para emular el trabajo con variables: Cumple, no se utiliza ninguna herramienta que difiere.
3. Todas las funciones debidamente comentadas: Se facilita un comentado para cada función explicando dominio y recorrido como llamados recursivos y sus explicaciones.
4. Respetar la definición de función en términos de conjunto de salida y llegada: Se respeta el dominio y recorrido en las funciones implementadas.
5. Se debe utilizar al menos dos tipos de recursividad en las siguientes funciones: Las funciones termQuery y phraseQuery contienen cada una de ellas dos tipos de recursividad distintos. El programa en general se trabaja con recursión natural y de cola.
6. Hacer uso de al menos dos funciones anónimas en una o más funciones: Las funciones termQuery y phraseQuery cumplen con el requisito.
7. Al final del código incluir tres ejemplos de uso para cada función: Se hace uso de al menos tres ejemplos para todas las funciones implementadas en el programa, cada línea en esta parte del código se encuentra comentada y explicada.
8. Proveer archivo con autoevaluación de las funciones: Se facilita este archivo junto con todo lo pedido en el enunciado.
9. Precargar los contenidos de los archivos de documentos y stopwords usando la estructura señalada en el enunciado: Cumple correctamente.
10. Implementación TDA: Se implementa una representación de un índice tanto como funciones de pertenencia, selectores y otras funciones. Las selectoras tienen problemas en algunos casos y los modificadores no lograron ser implementados.
11. createIndex: Función que cumple correctamente con lo que requiere, crea índices invertidos para distintos casos siempre y cuando los parámetros documentos y stopwords sean acordes con el criterio establecido.
12. termQuery: Cumple con lo especificado, creando una lista de resultados junto con el valor requerido por el enunciado, funciona para distintos casos.
13. phraseQuery: Función que cumple también con lo especificado. Se crea una lista de resultados mediante los criterios mencionados y junto al valor RankValue requerido.
14. ranking: Se aplica un algoritmo de búsqueda correctamente, la función recibe el respectivo resultado y entrega la lista ordenada, aun así, existen casos para el cual no se cumple.
15. Results->String: La funcion cumple con el requisito de mostrar de mejor manera los resultados al usuario, funciona para la mayoría de los casos sin problemas.

Las realizaron diferentes pruebas en el programa para cada una de las funciones implementadas y se facilita con el ejemplo al final del código.

Si bien, se pudieron implementar la mayoría de los requimientos funcionales obligatorios, se hace uso mucho de funciones que requieren mucho “peso”. Todas las funciones de Racket que son String-Split, String-Append, entra otras, requiere un uso muy grande de memoria que puede conllevar problemas o caídas en el programa. Sin embargo, se hace uso del paradigma funcional en la totalidad del programa, haciendo uso de recursión como principal herramienta como la composición de funciones.

Se crean funciones auxiliares para soluciones distintos problemas pequeños que surgieron durante el desarrollo del laboratorio. La representación elegida para el índice invertido se decide debido a los requerimientos pedidos en las funciones como phraseQuery o termQuery. La representación ayuda a solventar los problemas que surgen al realizar estas funciones.



# Conclusión

El objetivo principal de este laboratorio, es implementar un enunciado ciertos requerimientos mediante el paradigma de programación funcional. El objetivo fue cubierto debido a que se logró un desglose de análisis, diseño e implementación del programa mediante las reglas que rigen el paradigma funcional desarrollado y estudiado tanto en clases como de forma particular.

Hubo dificultades en el cambio del paradigma imperativo al funcional debido a que existen muchas herramientas en imperativo que no cuentan como reglas para la implementación del paradigma funcional. Sin embargo, se aprendió una nueva forma de abordar un problema. Como principal aprendizaje de este laboratorio fue la implementación de la recursión que en el caso de este programador en particular era muy pobre antes de entrar a este nuevo paradigma.

También se es consciente que el programa implementado puede mejorar mucho más su eficiencia con una mayor experiencia tanto en el paradigma como en el lenguaje. Esta mejoría se traduce en poder facilitar la lectura del código tanto como las funciones implementadas en él.

Se espera seguir trabajando arduamente bajo este paradigma para conseguir mejores resultados en la implementación de los programas.

# Referencias

Enunciado preliminar: <https://docs.google.com/document/d/1Fo3xcdk7BBd55uzeujcqeyY3LfAHy3RYRJPvhVyyIE/edit>

Seudocódigo ordenamiento burbuja: <http://4.bp.blogspot.com/-68E_CaKYtrw/TrwabX1MIEI/AAAAAAAAACQ/N4r_MINU0LU/s1600/algoritmo.png>

Requerimientos obligatorios informe: <http://www.udesantiagovirtual.cl/moodle2/mod/page/view.php?id=118922>