|  |
| --- |
| **PARADIGMAS DE PROGRAMACIÓN**  **LABORATORIO 3**  **PARADIGMA LOGICO** |
|  |
| **Realizado por: Jorge Luis Plaza Contreras** |

Profesor: Roberto González

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

Tabla de Contenidos

[Tabla de Contenidos I](#_Toc497849610)

[CAPÍTULO 1. Introducción 1](#_Toc497849611)

[1.1 Paradigma LOGICO 1](#_Toc497849612)

[1.2 Descripcion del problema 1](#_Toc497849613)

[CAPÍTULO 2. Descripción de la solución 2](#_Toc497849614)

[2.1 Analisis del problema 2](#_Toc497849615)

[2.2 Diseño de la solucion 3](#_Toc497849616)

[2.2.1 Representación de predicados índice, documento y stopword 3](#_Toc497849617)

[2.2.3 singleTermQuery 4](#_Toc497849618)

[2.2.4 bestMatch 5](#_Toc497849619)

[2.2.5 exactMatch 5](#_Toc497849620)

[2.2.6 documents 6](#_Toc497849621)

[2.2.7 numDocuments 6](#_Toc497849622)

[2.3 Aspectos de implementación 6](#_Toc497849623)

[CAPÍTULO 3. Análisis de resultados 7](#_Toc497849624)

[CAPÍTULO 4. Conclusión 8](#_Toc497849626)

[CAPÍTULO 5. Referencias 9](#_Toc497849627)

# Introducción

Durante el semestre se trabajara 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 LOGICO

El paradigma lógico se basa principalmente en los conceptos de la lógica matemática, donde las sentencias verdaderas y falsas son los protagonistas del paradigma.

Los programas desarrollados bajo este paradigma están organizados mediante una colección de sentencias expresadas como lógica. El programa se encarga, a través de las sentencias, de buscar una solución al problema planteado.

Este paradigma será implementado mediante el lenguaje Prolog y en el entorno SWI-Prolog.

Descripcion del problema

El programa debe ser capaz de responder a consultas realizadas por el usuario. Estas consultas vienen dadas por documentos, un índice y stopwords, estos últimos son las palabras no deseadas. Estas estructuras deben estar implementadas como una base de conocimiento para Prolog.

Con respecto a las funcionalidades del programa, deberá proporcionar el poder realizar consultas mediante una simple palabra o mediante una frase.

Debe ser capaz de responder si una frase ingresada por el usuario coincide totalmente con las palabras inmersas en un documento.

El caso viceversa, si se ingresa una lista de resultados, el programa debe ser capaz de entregar los títulos de los documentos correspondientes.

Finalmente, deberá poder entregar el número de documentos inmersos en el programa.

# 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, el programa debe ser implementado en el lenguaje de programación Prolog como se menciona en el informe. Se debe especificar los distintos conceptos inmersos en el programa, esto se hace mediante comentarios escritos en el mismo programa. Finalmente, se debe mantener un buen orden y organización del código correspondiente.

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

* Se debe expresar el índice, documentos y StopWords como una base de conocimiento a partir de los predicados índices, documento y stopword. Para realizar esto se necesita una representación correcta del programa, para realizarlo se necesita hacer un análisis completo de todos los requerimientos para poder encontrar una forma óptima para resolverlos.
* singleTermQuery: Este predicado permite al usuario realizar una consulta a través de una variable Term, el programa debería responder a través de Results, los documentos (IDs) que hacen que singleTermQuery sea verdadera. Para realizar este requerimiento es necesario que el predicado índice contenga cada término en los documentos juntos con IDs. De esta forma, al ingresar una consulta Term, el programa buscara dentro del predicado índice para responder al usuario correctamente.
* bestMatch: Este predicado determina si existe un documento o más que contengan uno o más términos de una consulta de tipo frase (Phrase) dada. Esta variable es una lista. Por lo tanto servirá para que el usuario pueda realizar una consulta mediante una frase. Para realizar este requerimiento se requiere escribir Phrase mediante una lista para trabajar de forma recursiva y mediante el predicado índice se buscaran el match correspondiente a la consulta de tipo frase. Para realizar la recursividad se necesita de un caso base escrito como hecho.
* exactMatch: Predicado a través del cual se puede determinar si existe un documento o más que contengan todos los términos de una consulta frase. Este predicado es similar al anterior pero esta vez de requiere que el documento tenga todas las palabras presentes en la consulta. Para realizarlo se necesita trabajar recursivamente mediante lista, por lo que, se necesita de un caso base escrito como hecho y de un buen trabajo de la lógica.
* documents: Predicado a través del cual se puede determinar si la relación entre una lista de documentos proporcionadas para una lista de resultados es consistente, entregando los títulos de los documentos. Para este predicado se requiere de una representación de los documentos acorde con lo requerido. Se requiere tener el titulo aparte en la representación para poder entregar el resultado correcto. Se necesita trabajar recursivamente para obtener todos los títulos que requiera el usuario.
* numDocuments: Predicado a través del cual se puede determinar si la cantidad de documentos especificados en Count y la cantidad de documentos contenidos en la lista de resultados corresponden. Para realizar se hace uso de recursividad, mediante un “contador” para lograr lo requerido.

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 Representación de predicados índice, documento y stopword

Gracias al desglose y completo análisis del problema, se llega a una representación de los predicados que ayudan a abordar los requerimientos de una forma óptima y coherente con los datos ingresados.

Para el caso del predicado stopword, se elige una representación simple que contiene solo un átomo, por lo tanto, el hecho stopword(a) está dando como verdadero que el átomo “a” es un stopword. El programa tendrá un conjunto completo de distintos stopword.

*Figura 1: Ejemplo stopword*

Del análisis del problema se infiere que para el predicado documento se necesita de un identificador del documento que viene dado por un simple número, se necesita del título del documento requerido por distintas funcionalidades obligatorias. Y por último, se necesita del texto del documento para obtener una mayor variedad de palabras en el índice. Cabe destacar que el título y el texto se ingresan como String para hacer que el predicado sea legible para el usuario.

*Figura 2: Ejemplo documento*

Para el último sub-problema de la elección de la representación se tiene el índice. Para este caso, se elige una representación con 2 datos. Uno de ellos es un átomo que representa una palabra dentro de un documento y el segundo es una lista de números, cada número representa el identificador del documento donde se encuentra la palabra. Esta representación facilita la búsqueda de las consultas y es fácil de organizar y ordenar. La figura 3, quiere decir que palabra se encuentra en los documentos 1, 2 y 3 respectivamente.

*Figura 3: Ejemplo índice*

### 2.2.3 singleTermQuery

Este predicado, mediante la representación escogida, es fácil de abordar. Se hace mediante una cláusula de Horn. La cláusula tiene dos términos, Term y Result. Por lo tanto, primero se verifica si la palabra ingresada por el usuario no es un stopword, mediante un not y también se tiene que cumplir, mediante índice, que la palabra coincide con Result. Por lo tanto, si el usuario ingresa un término que se encuentra en el documento, prolog mostrara la lista con los identificadores correspondientes. Si la palabra ingresada es un stopword o no se encuentra en el índice, prolog mostrara false.

### 2.2.4 bestMatch

Para realizar este predicado, se realizara recursividad. Para esto se necesita de un caso base que se escribirá como un hecho, si se ingresa una lista vacía como consulta, entonces coincide con otra lista vacía, dando a entender que no existe documento con una palabra que sea “vacía”.

Ahora, para realizar la cláusula, se utiliza de la representación de lista de prolog, [X|Xs] en este caso. Donde X representa la cabeza de la lista y Xs la cola de la lista. Así el programa, para cada llamado va a validar si es un stopword y mediante índice, se mostrara la lista de documentos que lo contienen. Luego se llama recursivamente con la cola de la lista. Como el predicado no requiere que los documentos no tengan todas las palabras de la consulta, se utiliza un OR (“;”).

Al realizar la consulta, prolog mostrara los resultados de cada palabra de forma separada, colocando un “;” para continuar con la consulta.

### 2.2.5 exactMatch

exactMatch funciona de una forma muy parecida a bestMatch, por lo tanto, se aplica el método de resolución de problema similitud débil.

Para este predicado, se requiere de dos listas. Por lo tanto, se escriben como [X|Xs] y [D|Ds] respectivamente. La recursividad se aplica igual que en bestMatch solo que se pasan dos listas y para que el documento contenga todas las palabras de la consulta se necesita del operador AND (“.”). Finalmente prolog muestra los documentos mediante una lista.

### 2.2.6 documents

Nuevamente se requiere aplicar recursividad para realizar este predicado. El caso base se mantiene igual, si se ingresa una lista vacía se muestra una lista vacía.

Como la lista de resultados, es una lista de identificadores de documento. Se necesita que el predicado documento sea acorde con cada identificador ingresado. Por lo tanto se aplica este predicado y cada título se guarda como cabeza de la lista anotada como [Y|Ys] en este caso. La variable texto no se requiere para este predicado, por lo tanto, solo se coloca como auxiliar para evitar problemas de compilación.

Luego mediante recursividad, se aplica documents nuevamente pasando ambas colas de las listas.

### 2.2.7 numDocuments

Para este predicado se necesita trabajar recursivamente aplicando una variable para sumar por cada documento inmerso en la lista ingresada. Nuevamente se usa como notación [X|Xs] para poder trabajar tranquilamente sobre la lista, luego mediante el uso de la notación AND y una variable Caux se calcular el número de documentos.

## 2.3 Aspectos de implementación

Se implementaron todos los requerimientos obligatorios junto a dos extras. Al programa se le implementaron los más de 300 stopwords inmersos en el archivo de usach virtual.

Para mantener un programa más compacto, se utilizaron los primeros 5 documentos del archivo TextCollection. Sin embargo, esto no influye en el desarrollo de los predicados implementados.

# Análisis de resultados

Los requerimientos obligatorios del proyecto son los siguientes:

1. Implementación en lenguaje de programación Prolog: Cumple totalmente con el requisito.
2. Se debe documentar el código indicando dominios, predicados, metas y cláusulas de Horn: Se documenta todo lo pedido.
3. Organización del código (orden y claridad): El código mantiene un orden adecuado para el entendimiento de un programador.
4. Expresar el índice, documentos y StopWords como una base de conocimiento a partir de los predicados índice, documento y stopword: Mediante una representación que es justificada en el informe, se implementan los 3 predicados pedidos.
5. singleTermQuery: El predicado funciona correctamente realizando lo pedido en el enunciado.
6. bestMatch: Predicado cumple con lo pedido en el enunciado.
7. exactMatch: El predicado tiene ciertos problemas y para algunos casos cumple con lo especificado. Pero a veces solo da los IDs de los documentos de cada palabra y no solo los IDs en los que se cumple.
8. Documents: Predicado cumple con lo pedido y funciona correctamente.
9. numDocuments: Predicado cumple correctamente con lo pedido sin problemas.

El código queda bastante claro en su cometido implementando los predicados obligatorios requeridos. Prolog lanza un mensaje de precaución para dos variables que no se utilizan pero son necesarias para no tener problemas de compilación. También se implementaron los extras termTotalCount y maxFrequency logrando cumplir con sus requisitos.



# Conclusión

El objetivo principal de este laboratorio, es implementar un enunciado ciertos requerimientos mediante el paradigma de programación lógico. 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 lógico desarrollado y estudiado tanto en clases como de forma particular.

Hubo dificultades en el cambio del paradigma imperativo y funcional al lógico debido a que existen muchas herramientas en los otros paradigmas que no cuentan como reglas para la implementación del paradigma lógico. Sin embargo, se aprendió una nueva forma de abordar un problema. Como principal aprendizaje de este laboratorio fue la implementación de la lógica.

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 y eficiencia del código con respecto a sus predicados.

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>

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