|  |
| --- |
| **PARADIGMAS DE PROGRAMACIÓN**  **LABORATORIO 1**  **PARADIGMA IMPERATIVO** |
|  |
| **Realizado por: Jorge Plaza C** |

Profesor: Roberto González

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

Tabla de Contenidos

[Tabla de Contenidos I](#_Toc492393489)

[Índice de Figuras I](#_Toc492393490)

[CAPÍTULO 1. Introducción 1](#_Toc492393491)

[1.1 Paradigma imperativo 1](#_Toc492393492)

[1.2 Descripcion del problema 1](#_Toc492393493)

[CAPÍTULO 2. Descripción de la solución 2](#_Toc492393494)

[2.1 Analisis del problema 2](#_Toc492393495)

[2.2 Diseño de la solucion 3](#_Toc492393496)

[2.2.1 Leer y guardar archivos 3](#_Toc492393497)

[2.2.2 Creación lista de palabras 3](#_Toc492393498)

[2.2.3 Crear matriz de incidencia 4](#_Toc492393499)

[2.2.4 Guardar y cargar índice 4](#_Toc492393500)

[2.2.5 Consulta y ranking 5](#_Toc492393501)

[2.2.6 Mostrar resultados 5](#_Toc492393502)

[2.3 Aspectos de implementación 5](#_Toc492393503)

[CAPÍTULO 3. Análisis de resultados 7](#_Toc492393504)

[CAPÍTULO 4. Conclusión 9](#_Toc492393506)

[CAPÍTULO 5. Referencias 10](#_Toc492393507)

Índice de Figuras

[Figura 2‑1: Ejemplo de matriz de incidencia 4](#_Toc382443674)

[Figura 2‑2: Seudocodigo ordenamiento burbuja 5](#_Toc382443675)

[Figura 2‑3: Esquema de las funciones y sus relaciones 6](#_Toc382443676)

# 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 imperativo

La programación imperativa está basada en el uso de variables y comandos que modifican el estado de estas. Un conjunto de comandos se conoce como procedimiento.

Para la implementación de este paradigma se utiliza el lenguaje de programación C. Incluyendo herramientas que permiten administrar la memoria.

Descripcion del problema

El programa, en términos generales, debe leer archivos de texto como 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 y mostrar los M primeros resultados al usuario.

El segundo archivo de texto contiene una lista con StopWords, es decir, palabras que no se deben considerar al momento de crear el índice o realizar la consulta.

También, tiene que ser capaz de guardar el índice de los documentos en un archivo de texto, para posteriormente, cargar el índice nuevamente y ejecutar una búsqueda mediante la consulta.

# Descripción de la solución

2.1 Analisis del problema

Para abordar el problema, se necesita hacer un análisis de los puntos más importantes que se necesitan realizar para lograr una buena solución.

* Se requiere leer el archivo de StopWords y guardarlos en una estructura
* Se requiere leer el archivo de los documentos y guardarlos en una estructura.
* Se necesita obtener una lista con todas las palabras en el documento.
* De esta lista se requiere eliminar las palabras StopWords obtenidas inicialmente.
* De la misma lista, se requiere eliminar todas las palabras duplicadas.
* Con la lista de las palabras y los documentos, se requiere crear una matriz de enteros.
* Esta matriz contendrá en cada espacio, la cantidad de incidencias de la palabra en el documento.
* Se requiere escribir el índice en un archivo de texto.
* Mediante un identificador, se debe cargar un índice.
* Se necesita realizar una consulta y un ranking.
* El ranking debe ser ordenado mediante la cantidad de repeticiones de la consulta.
* Finalmente, mostrar los resultados al usuario.

Estos puntos, son esenciales para lograr una buena solución. También pueden existir otros que son opcionales para la implementación del programa.

2.2 Diseño de la solucion

Debido a la extensión del problema, se abordara mediante el método de resolución llamado división en sub-problema. Donde cada uno de estos problemas corresponden a los puntos mencionados en el análisis del problema.

### 2.2.1 Leer y guardar archivos

Para leer y guardar los archivos de texto, se necesita de dos estructuras importantes en el programa. Estas estructuras llevan los nombres de StopWords e Index. Para poder guardar los textos, se necesitan estructuras que guarden cadenas de caracteres. En el caso de StopWords, se utiliza un array doble char[][] y un entero que guarda la cantidad de palabras en el archivo. Para la realización del índice, se necesitan tres variables sumamente importantes. Una de ellas corresponde a un puntero a otra estructura, que se llama Result y corresponde a un documento del archivo. Esta estructura guarda todo lo que se necesita de un documento, titulo, autor, extracto, lista de palabras, un identificador, cantidad de palabras para cada una de las variables, entre otras. Entonces, la estructura Result en el índice guarda en cada posición de la memoria un documento distinto.

Por otra parte, el índice guarda un doble puntero a entero. Esta estructura permitirá saber la cantidad de repeticiones de las palabras en cada documento, donde las filas corresponden a la palabra y las columnas al documento.

Finalmente, se necesita una lista total de las palabras encontradas en el documento, esta lista excluye StopWords y duplicados.

### 2.2.2 Creación lista de palabras

Para crear la lista de palabras, se implementan tres funciones. Que se llaman obtenerListaPalabras, eliminarDuplicados y eliminarPalabra. La primera de ellas, mediante el índice y StopWords, obtiene cada palabra del archivo y para cada palabra hace un llamado a eliminarPalabra, la cual, mediante un algoritmo simple, retorna 0 o 1 dependiendo si la palabra es un StopWord. Al final de la funcion obtenerListaPalabras, se llama a eliminarDuplicados que se encarga de eliminar las palabras repetidas del listado. Para finalmente obtener la lista final que queda almacenada en el índice.

### 2.2.3 Crear matriz de incidencia

Para la creación de esta estructura, se crea una función que se llama crearMatriz. Esta función se llama luego de obtener la lista final en la función eliminarDuplicados. Para crearla se utilizan for anidados, cada uno de estos for tiene el largo de las estructuras en el índice, el largo de la lista y el largo de los documentos, luego, cada vez que se encuentre una palabra en el documento se le suma a la posición. Por ejemplo, si la palabra “Investigation” que está en la primera posición de la lista final se encuentra 3 veces en el documento 1, entonces, la posición de la matriz [0][0] será igual a 3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Documento 1 | Documento 2 | Documento 3 | Documento 4 | Documento 5 |
| Investigation | 3 | 0 | 1 | 2 | 4 |
| Experimental | 1 | 2 | 0 | 0 | 1 |
| Boundary | 1 | 0 | 0 | 0 | 1 |
| Layer | 2 | 2 | 1 | 0 | 0 |

Figura 2-1: Ejemplo de matriz de incidencia

### 2.2.4 Guardar y cargar índice

Para guardar el índice se necesita un identificador, el cual, se coloca en la primera línea del documento, después del primer “.I”, así se podrá leer y cargar el índice posteriormente. Para guardarlo, se utilizan solo ciclos for. Cada uno de ellos escribe las líneas correspondientes al título, autor, fecha o lugar y el extracto.

Para cargar el índice, se necesita cargar nuevamente los StopWords y luego a la función que guarda el índice. Esto solo ocurre si el identificador ingresado es el correcto.

### 2.2.5 Consulta y ranking

La consulta que realiza el usuario se hace en la función main del programa, esta consulta se guarda y se envía a la función query que se encarga de crear el ranking. Para el ranking se utiliza una estructura, esta estructura contiene una lista de documentos que es ordenada mediante un parámetro llamado repeticionesPalabra. Este parámetro se utiliza en cada documento y lo que hace es aumentar su valor cada vez que las palabras en la consulta se encuentran en la matriz de incidencia, solo si, su valor es distinto de 0. Luego, mediante este valor, se utiliza un algoritmo de ordenamiento llamado burbuja, que ordena los documentos de mayor a menor dependiendo de este valor.

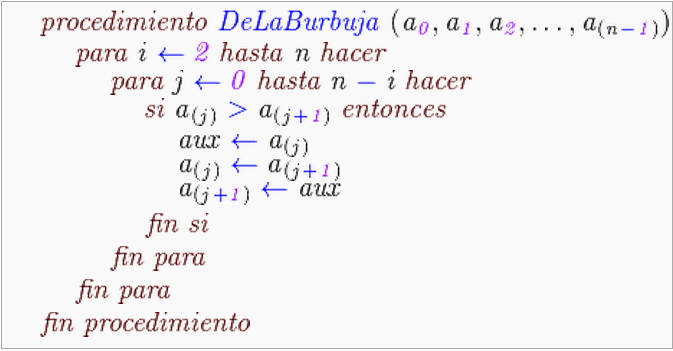


Figura 2-2: Seudocódigo del ordenamiento burbuja

### 2.2.6 Mostrar resultados

Finalmente, se muestra los resultados al usuario mediante la función printf y la lista de documentos que se encuentran en el ranking, dependiendo de la cantidad de resultados que quieren ser mostrados.

## 2.3 Aspectos de implementación

El programa está separado en 4 cuatros archivos, estos son main.c, funciones.c, extras.c y cabecera.h. El archivo cabecera.h, contiene las 4 librerias usadas, en este caso, stdio.h, stdlib.h, string.h y time.h. Estas librerías fueron escogidas debido a que facilitan el trabajo con cadena de caracteres. El archivo también contiene la variable enum, las estructuras del programa y las declaraciones de todas las funciones presentes. El archivo main.c contiene la funcion principal del programa, que contiene un menú para probar las distintas funcionalidades y que finalmente libera la memoria. El archivo funciones.c contiene todas las funciones del programa con sus algoritmos correspondientes y el archivo extras.c contiene funciones opcionales del programa.

El compilador utilizado es gcc y también se facilita un archivo makefile para lograr una compilación correcta del programa.

Figura 2-3: Diagrama de las funciones y sus relaciones

# Análisis de resultados

Los requerimientos obligatorios del proyecto son los siguientes:

1. Implementación en lenguaje C: Cumple totalmente con el requisito.
2. Solo recursos de ANSI C: Cumple, no se utiliza ninguna herramienta que difiere.
3. Garantiza compilación en Windows y Linux: Si se sigue con las instrucciones de compilación, se cumple.
4. Utilizar formato de salto de línea de Unix, basado en ‘\n’: Cumple totalmente.
5. Se debe trabajar con archivos texto plano: Cumple correctamente.
6. Implementación debe ser lo más genérica posible: La implementación es genérica si se cumple con las reglas de entrada. Por ejemplo, el archivo de texto con los documentos tiene que venir con el formato “.T”, “.A”, “.B” para sus distintas secciones.
7. Se deben validar entradas a fin de controlar errores en la lectura: El programa muestra errores si los directorios de los archivos de texto son incorrectos.
8. Organización del código mediante archivos .h y .c: Cumple correctamente.
9. Makefile para la correcta compilación del proyecto: Se facilita un archivo makefile para la compilación.
10. Se debe respetar las interfaces definidas en el enunciado: Cumple en cada una de las funciones.
11. De deben utilizar las estructuras del enunciado: Se utilizan las mismas estructuras del enunciado.
12. Para efectos de reportar el estado utiliza enum code: Se utiliza esta estructura para reportar estados de las funciones.
13. loadStopWords: La funcion permite leer el archivo de texto y almacena las palabras en la estructura StopWords.
14. createIndex: Permite la creación del indice, se complementa con cuatro funciones creadas por el programador incorporando el conjunto de StopWords.
15. saveIndex: La función autogenera un identificador y se usa para guardar el índice correspondiente. También imprime la fecha en que fue guardado y su identificador.
16. loadIndex: Recibe el identificador de un índice y lo retorna mediante las funciones creadas anteriormente.
17. Query: Utiliza la consulta de usuario y genera un ranking con los documentos de mayor a menor tomando en cuenta la lista de StopWords. En caso de obtener documentos sin las palabras consultadas, ordena los documentos según su número en el archivo y los imprime dependiendo de la cantidad de resultados que quieren ser mostrados.
18. displayResults: La función permite mostrar los Top K resultados obtenidos en query.
19. Menu en main: Se facilita un menú simple en la función que permite probar las funcionalidades del programa.

Se realizaron pruebas para probar el funcionamiento del programa, ingresando distintas consultas y archivos de texto. En general, el programa logra correctamente leer archivos para distintas entradas y guardarlos posteriormente en archivos de texto. Para las consultas, también se realizaron muchas pruebas obteniendo resultados satisfactorios y otros no tanto. Con algunas consultas, por ejemplo “boundary layer”, el programa no coloca el documento con mayores incidencias en la parte más alta, sino en una de las posiciones inferiores. Se asume que este error se debe al algoritmo de ordenamiento o a la matriz de incidencia.

Mencionar, que el programa tiene que cargar los StopWords antes de guardar el índice o que necesita guardar el índice antes de realizar una consulta. Cualquiera de estos casos puede causar la caída del programa. Esto se debe a que cada función requiere una entrada que no sea nula.

El programa, si bien administra en ciertas partes la memoria, no lo hace completamente. Por lo tanto, una entrada muy grande en los archivos de texto puede conllevar a una caída del proyecto.

Decir nuevamente que el programa necesita tener los directorios de manera correcta de los archivos de texto. Hay 4, dos en el main con los nombres directorioStopWords y directorioDocumentos y dos en la funcion loadIndex, uno llamado directorioIndiceAuxiliar y otro que se llama desde la funcion loadStopWords. Estos directorios corresponden, como sus nombres lo dicen, al del archivo de StopWords y el archivo con los textos. En caso de no tener los directorios correctamente, mostrara un mensaje de error.



# Conclusión

El programa logra los requerimientos obligatorios pero no se implementa un algoritmo general óptimo para su causa, es decir, el tiempo de ejecución podría simplificarse mucho para el objetivo que realiza. En otras palabras, el programa realiza muchos ciclos gigantescos y anidados que complican la compilación y ejecución, como también, tiene mucho código extra.

También, se limita mucho los archivos de texto como entrada. Un archivo muy gigantesco, como por ejemplo, 1000 documentos, no lograra compilar correctamente debido a que en esta parte el programa no administra la memoria, más bien lo hace de forma fija. Es decir, no aceptara un archivo de texto con más 16384 palabras en él.

Aun así, como se menciona, se cumple con todo lo requerido por el enunciado del informe, incluyendo todas las estructuras y funciones obligatorias puestas en él. Se trabajara posteriormente de forma independiente para optimizar el rendimiento del programa.

# 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>