ANÁLISIS Y DISEÑO DE ALGORITMOS I

2do Cuatrimestre 2020

Trabajo Práctico Especial

Plataforma eCommerce de Libros Históricos

Grupo 17, Profesor: Diego Tolbaños.

Alumnos: Bricio Vellaz Barbieri. Email: briciovellaz@gmail.com
Hilen Vicenzi. Email: hilenvicenzi99@gmail.com

ÍNDICE

1. Resumen

El siguiente informe pertenece al trabajo practico especial de la cátedra Análisis y diseño de algoritmos I.

Como objetivo, tenemos que desarrollar un programa en C++ que permita resolver el problema que se nos plantea.

Este problema trata de que una plataforma *eCommerce*, que contiene una extensa base de datos de libros históricos, necesita un sistema de consultas para los clientes. Se nos pide que el sistema lleve un registro de los libros con los que cuenta la plataforma y proporcionar a los clientes distintos servicios de búsqueda para poder responder a las consultas que se realicen.

A lo largo del informe vamos a explicar:

- el objetivo de la creación de este programa,
- el diseño que implementamos para crear las diferentes *TDA* que les dan forma a los contenedores utilizados para almacenar libros y poder resolver los servicios,
- la implementación del código donde desarrollaremos la explicación del código final así también como algunos aspectos del código fuente, facilitado por la cátedra, que fuimos modificando a lo largo del desarrollo de este programa y
- Hablaremos sobre la complejidad temporal de todos los procedimientos.

2. Objetivo y descripción del problema

El objetivo de este trabajo es realizar un programa que cuente con los servicios de búsqueda pedidos y tener un buen registro de los libros con los que cuenta la plataforma. Este programa será ejecutado una única vez en el día y se detendrá al finalizar la jornada, por lo tanto, los servicios de búsqueda se deberán resolver un número considerable de veces.

Utilizaremos un *TDA Lista* que será de un tipo *TDA Libro* que almacenará todos los libros que se encuentren en un archivo de texto, lo cual se explicará a lo largo del informe.

Los servicios que el programa debe contener son 3, y todos son servicios de búsqueda. Esto quiere decir que dado uno varios parámetros ingresados por el cliente se busca un/unos libro/s que tengan las mismas características que el buscado. Los servicios consisten en:

- Búsqueda de un libro dado su título, con la posibilidad de usar dos comodines diferentes. El signo de pregunta '?' sería utilizado para reemplazar solo un carácter del título. El signo asterisco '*' sería utilizado para reemplazar una palabra del título.
- Búsqueda de todos los libros que se encuentran en un determinado rango de fechas, mostrando toda su información incluyendo la lista de idiomas en los que se encuentran disponibles.
- Búsqueda de los libros más económicos que se encuentra en un determinado idioma, mostrando hasta 10 (diez) de ellos.

A lo largo del código se puede ver que empleamos librerías propias de C++ además de las clases construidas para poder resolver los problemas a resolver.

3. Diseño

3.1 TDA: Tipos de datos abstractos

Mientras analizábamos el problema surgió el reconocimiento de un TDA en específico que se vio durante las clases teóricas y prácticas de la cátedra. Nos ayudamos mucho del material que se fue brindando a lo largo del cuatrimestre, específicamente del archivo "Especificación algebraica de tipos de datos abstractos: el lenguaje Nereus" aparte de foros y páginas en las que fuimos encontrando información.

Una vez visto el enunciado y analizado las diferentes maneras en las que se podría resolver el problema llegamos a la conclusión de que para poder almacenar los distintos libros con los que cuenta el archivo de texto tenemos que diseñar un contenedor que a su vez nos ayude a la hora de hacer cualquier tipo de búsqueda. En nuestro caso decidimos implementar un *TDA Lista*. Este consiste en diferentes procedimientos básicos como la creación de una lista, el agregar un elemento, avanzar dentro de la lista, recuperar un elemento/nodo, mostrar por pantalla el contenido de un nodo determinado, etc

A su vez también decidimos implementar un *TDA Libro* para que este tipo de dato sea almacenado en una lista. Consiste en procedimientos básicos que devuelven una variable y un procedimiento para agregar a este tipo de datos una lista, que en nuestro caso sería una lista de idiomas en los que se encuentra disponible el libro.

3.2 Clase Libro

La plataforma con la que vamos a trabajar cuenta con miles de libros, y cada uno de ellos tiene que almacenar una información determinada. Esta información consiste en:

- ISBN del libro (International Standard Book Number)
- Título del libro
- Autor del libro
- Año de publicación del libro
- País de origen
- Lista de Idiomas en los que está disponible
- Precio del libro en dólares

Por lo tanto, la clase libro tendrá que contener variables que almacenen cada uno de los puntos mencionados anteriormente. La clase libro va a contar con las siguientes variables: string isbn; string titulo; string autor; string fechaPublicacion; string paisOrigen; float precio.

A su vez vamos a usar una estructura Lista que va a ser de tipo string, ósea que va a contener en cada uno de los nodos cadenas de caracteres. En nuestro caso van a ser los idiomas en los que el libro se encuentra disponible. Esta lista se va a llamar idiomas.

Las funciones que la clase va a contener son:

- Libro()

Es el constructor de la clase

Complejidad: O(1)

- ~Libro()

Es el destructor de la clase

Complejidad: O(1)

- Libro(string isbn, string titulo, string autor, string fechaPublicacion, string paisOrigen, float precio)

Método constructor de la clase, asigna a cada una de las variables su respectivo valor.

Complejidad: O(1)

- string devolverIsbn()const

Método observador de la clase, retorna el valor de Isbn

Complejidad: O(1)

- string devolverTitulo()const

Método observador de la clase, retorna el valor de Titulo

Complejidad: O(1)

- string devolverAutor()const;

Método observador de la clase, retorna el valor de Autor

Complejidad: O(1)

- string devolverFechaPublicacion()const

Método observador de la clase, retorna el valor de FechaPublicacion

Complejidad: O(1)

- string devolverPaisOrigen()const

Método observador de la clase, retorna el valor de PaisOrigen

Complejidad: O(1)

- float devolverPrecio()const

Método observador de la clase, retorna el valor de Precio

Complejidad: O(1)

- Lista<string> & devolverIdiomas()

Método observador de la clase, retorna el valor de Idiomas

Complejidad: O(1)

- void agg_idioma (string idiom)

Método constructor de la clase, agrega un idioma a la lista de string.

Complejidad: O(1)

A su vez dentro de esta clase (Libro) vamos a encontrar esto:

ostream & operator << (ostream & salida, Libro & l)

Con este procedimiento vamos a poder mostrar por pantalla todas las variables que presenta un libro, incluyendo los idiomas separándolos por coma y mostrándolos uno a uno.

3.3 Clase Lista

Decidimos crear un TDA Lista para que este sea nuestro contenedor.

La lista es una estructura dinámica de datos que contiene una cantidad determinada de elementos de un mismo tipo, lo que la vuelve homogénea y de cierta manera se puede establecer un orden entre ellos.

En nuestro caso a medida que se lee una línea del archivo, que contiene a los libros que se deben cargar en la plataforma, va creando un nodo y agregándolo a la lista. En el archivo se puede ver que los libros están ordenados alfabéticamente, pero de la manera en la que decidimos resolver la inserción de nodos de la lista, esta va a quedar ordenada de forma descendente.

Para crear esta clase usamos una técnica de programación conocida como metaprogramacion (metaprogramming). Esta técnica utiliza plantillas (templates) para generar código fuente temporal, fusionándolo con el resto de código fuente y compilándolo. Usando esta técnica evitamos crear varias clases con el mismo código para diferentes tipos de datos, ya que al usar plantillas de las clases facilitan la reutilización del mismo código permitiendo que se usen para todo tipo de datos.

La clase lista contiene:

struct nodo{

T elem;

*nodo * sig;*

};

Esta es la estructura que tiene nuestro nodo. T elem es lo que va a contener el nodo (cualquier tipo de variable) y un nodo que va a apuntar al siguiente de la lista.

*nodo * primero;*

Siempre va a apuntar al primer nodo de la lista.

*nodo * cursor;*

Es el nodo de usaremos para movernos de nodo en nodo.

*nodo * crear_nodo (const T & elem, nodo * sig);*

Un nodo que va a apuntar al nodo que se va a crear.

unsigned int cant_elem = 0;

Variable que usaremos como contador de nodos de la lista. Lo usaremos para saber cual es la longitud de la lista

Las funciones que la clase va a contener son:

- Lista()

Método constructor de la clase. Asigna al nodo 'primero' el valor null y asigna a cant_elem el valor 0.

Complejidad: O(1)

- ~Lista()

Destructor de la clase. Está vacío.

- vaciar()

Método modificador de la clase. Vacia por completo la lista.

Complejidad: O(n) con n: longitud de la lista

- crear_nodo (const T & elem, nodo * sig)

Método constructor de la clase. Crea un nuevo nodo.

Complejidad: O(1)

- agregar_elem(const T & nuevo_elem)

Método modificador de la clase. Agrega un nodo, previamente creado, a la lista. Llama a crear_nodo.

Complejidad: O(1)

- mostrar() const

Método observador de la clase. Muestra por pantalla el elemento al que el cursor está apuntando.

Complejidad: O(1)

- longitud_lista () const

Método observador de la clase. Devuelve la variable cant_elem.

Complejidad: O(1)

- cursor_principio()

Método modificador de la clase. Apunta el cursor al primer nodo.

Complejidad: O(1)

- recuperar_elem () const

Método observador de la clase. Retorna el elemento que el cursor está apuntando.

Complejidad: O(1)

- avanzar_cursor ()

Método modificador de la clase. Hace apuntar el cursor al nodo que sigue.

Complejidad: O(1)

3.4 Clase Librería

Decidimos implementar una clase a la que llamamos Librería donde tendrá todos los servicios de la plataforma de libros junto con sus funciones auxiliares.

De esta manera el código que mucho mas prolijo y modularizado, dejando a la vista del usuario solamente las tres funciones principales.

Dentro la parte privada, la clase Librería contiene:

Variables:

- Lista<Libro> libros;
- Libro aux;
- Strings:
  - Titulo,
  - fecha_usuario_1,
  - fecha_usuario_2,
  - idioma,
  - nro_libros[100],
  - p,
  - b.
- int:
  - i;
  - cant_usuario;

Funciones auxiliares:

- void sinacentos(string &p);

Quita acentos y las ñ de cualquier string.

Complejidad: O(n) siendo n la longitud del string.

- bool compararSigno(string p, string b);

Compara el string sacado de la lista con el string ingresado por el usuario junto con el comodín elegido, en este caso la función corresponde al comodín '?'

Complejidad: O(n) siendo n la longitud del string ingresado por el usuario.

- bool compararAsterisco(string p, string b);

Compara el string sacado de la lista con el string ingresado por el usuario junto con el comodín elegido, en este caso la función corresponde al comodín '*'

Complejidad: O(n) siendo n la longitud del string ingresado por el usuario.

- int cant_libros(Lista<Libro> & libros, string idioma);

Retorna la cantidad de libros que hay en la lista de libros que tengan el mismo idioma.

Complejidad: O(n^2) siendo n la longitud de la lista de libros y la longitud de la lista de idiomas.

- bool mismo_idioma(Lista<Libro> & libros, string idioma, Libro aux);

Retorna una función bool que nos dice si el idioma de un libro dentro de la lista es igual al idioma ingresado por el ususario.

Complejidad: O(n) siendo n la longitud de la lista de idiomas.

- float busqueda_precio (Lista<Libro> & libros, string nro_libros[], int i);

Retorna el precio de un libro dado si isbn que se encuentra guardado de un arreglo de strings.

Complejidad: O(n) siendo n la longitud de la lista de lirbos.

- void ordenar_arreglo(Lista<Libro> & libros, string idioma, string nro_libros[]);

Ordena el arreglo (donde se guardan los isbn de ciertos libros) dado un precio que es dado por la función *búsqueda_precio*.

Complejidad: O(n^2) siendo n la longitud del arreglo que fue dado por la función *cant_libros*.

*O(n^2) + [O(n)*O(n)] + 2O(n) → O(n^2) + O(n^2) + 2O(n) → 2 O(n^2) + 2O(n)*

- void guardar (Lista<Libro> & libros, string idioma, string nro_libros[]);

Guarda en un arreglo de string los isbn de los libros que tiene dentro de su lista de idiomas el mismo idioma que ingreso el usuario.

Complejidad: O(n^2) siendo n la longitud de la lista de libros.

Dentro de la parte publica, en Librería se encuentran los 3 servicios principales:

- void existe_libro (Lista<Libro> & libros, string titulo);

Determina si existe un libros dentro de la lista dado un titulo ingresado por el usuario. Tiene la posibilidad de usar dos comodines distintos.

Complejidad: O(n) siendo n la longitud de la lista de libros.

- void listar_por_fecha (Lista<Libro> & libros, string fecha_usuario_1, string fecha_usuario_2);

Muestra por pantalla los libros que se encuentran entre dos fechas ingresadas por el usuario.

Complejidad: O(n) seindo n la longitud de la lista de libros.

- void mostrar_economicos (Lista<Libro> libros, string idioma, int cant_usuario);

Muestra por pantalla los N libros que desea ver el ususario en un determinado idioma que también es ingresado por el usuario.

Complejidad: O(n^3) siendo n la cantidad de libros que quiere ver el usuario y la longitud de la lista de libros.

3.5 Especificaciones en lenguaje NEREUS

3.5.1 TDA Lista

CLASS Lista [elem: T]

IMPORTS Libro, string,

BASIC CONSTRUCTORS Lista, agregar_elem

EFFECTIVE

TYPE Lista

OPERATIONS

Lista: → Lista;	
vaciar: Lista * Nat → Lista;	Modificadora
agregar_elem: Lista * elem → Lista;	Modificadora
mostrar: Lista → elem;	Observadora
longitud_lista: Lista → Nat;	Observadora
cursor_principio: Lista → Lista;	Modificadora
recuperar_elem: Lista → elem;	Observadora
avanzar_cursor: → Lista;	Modificadora

AXIOMS

...

END-CLASS

3.5.2 TDA Libro

CLASS Libro [string isbn, string titulo, string autor, string fechaPublicacion, string paisOrigen, float precio, Lista<string> idiomas]

IMPORTS Lista, string, float.

BASIC CONSTRUCTORS Libro,

EFFECTIVE

TYPE Libro

OPERATIONS

Libro: → Libro;	Constructora básica

devolverIsbn: Libro → string; Observadora

devolverTitulo: Libro → string; Observadora

devolverAutor: Libro → string; Observadora

devolverfechaPublicacion: Libro → string; Observadora

devolverpaisOrigen: Libro → string; Observadora

devolverPrecio: Libro → float; Observadora

devolverIdiomas: Libro → string; Observadora

Agg_idioma: string → Lista; Modificadora

AXIOMS

...

END-CLASS

3.5.3 TDA Librería

CLASS Librería [Lista<Libro> libros; Libro aux,; string titulo, fecha_usuario_1, fecha_usuario_2, idioma, nro_libros[100], p, b; int i, cant_usuario]

IMPORTS Lista, Libro, string, int.

BASIC CONSTRUCTORS Libreria

EFFECTIVE

TYPE Libro

OPERATIONS

Librería: →Libreria; Constructora básica

existe_libro: libros * titulo → libros; Observadora

listar_por_fecha: libros*fecha_usuario_1*fecha_usuario_2 → libros
Observadora

mostrar_economicos: libros*idioma*cant_usuario→libors Observadora

AXIOMS

...

END-CLASS

4. Implementación

Para poder crear los servicios que se nos pedían, implementamos las clases Libro y Lista que fueron explicadas anteriormente.

4.1 Función procesar_archivos_entrada

Esta función es el código fuente que la cátedra nos facilito para poder procesar el archivo de texto que contiene todos los libros que va a tener la plataforma eCommerce.

Fuimos modificándolo y agregando nuevas cosas para que funcionara. Esta función trabaja con la lista 'principal' (Lista<Libro> libros) del programa que va a ser el contenedor de todos los libros del archivo de texto.

Esta porción de código funciona de la siguiente manera: el archivo de texto está dividido por renglones y estos renglones están divididos por punto y comas. Esto quiere decir que cada vez que se encuentra con un punto y coma (;) se va a encontrar con una nueva variable que va a ser almacenada. Entonces:

- Lee la primera línea de código,
- Encuentra el primer dato, en este caso seria el ISBN que copia lo que encuentra en una variable string llamada isbn y para cuando llega a un punto y coma
- Encuentra el segundo dato, en este caso el título, lo copia en una variable string hasta encontrar un punto y coma
- Así sucesivamente con toda la información que tiene que tener el libro: isbn, titulo, autor, fecha de publicación, país de origen y precio.
- Crea una variable de tipo Libro llamada l donde almacena todos esos datos obtenidos del archivo de texto. Esta se crea mediante la función Libro de la clase Libro.

Una vez que fue creado un libro a este le falta agregar los idiomas en los que se encuentra disponible, a lo que nosotros hicimos el siguiente código:

De esta manera creamos la lista de idiomas que estará almacenada dentro del libro l:

```
pos_inicial = pos_final + 2;

pos_final = linea.size()-1;

string idiomas = linea.substr(pos_inicial, pos_final - pos_inicial);

int pos_inicial_coma = 0, pos_final_coma = 0;
```

Extrajimos la porción en la que se encuentran todos los idiomas y los almacenamos en una sola variable llamada idiomas. De ahora en adelante cada vez que se encuentra una coma separaremos el string para que a la hora de agregarlos en la lista cada nodo sea un solo string que contenga un solo idioma.

```
while (pos_final_coma != -1) {

  pos_final_coma = idiomas.find(',', pos_inicial_coma);

  string idioma = idiomas.substr(pos_inicial_coma, pos_final_coma - pos_inicial_coma);

  pos_inicial_coma = pos_final_coma + 1;
```

Usamos una función propia de c++ para que borre los espacio que hay entre coma y coma:

```
idioma.erase(std::remove(idioma.begin(), idioma.end(), ' '), idioma.end())
```

Agrega a la lista de idiomas el primer idioma encontrado

```
  l.agg_idioma(idioma);

}
```

Una vez que termino todo lo anterior, tanto la creación del Libro l como la creación de la lista de idiomas, se agrega a la *Lista<Libro> libros* en forma de nodo el *Libro l.*

La complejidad de esta función es de *O(n)* siendo n la cantidad de libros a agregar a la lista 'libros'

4.2 Servicio 1: Búsqueda de libro por titulo

El primer servicio que se nos pide realizar consiste en la búsqueda de un libro por su titulo y verificar si existe dentro de la plataforma. A su vez el usuario tiene la posibilidad de usar dos tipos de "comodines":

- Asterisco (*): es un símbolo que se puede usar como comodín a la hora de buscar un título. Si lo usamos este reemplazará a una sola palabra del título, ya sea porque no nos acordamos o queremos ver qué libros tiene esa palabra.
- Signo de pregunta ('?'): se puede usar como comodín para reemplazar un solo carácter del título.

Dentro de este servicio usamos otras funciones para facilitarnos la búsqueda del libro:

- Sinacentos: reemplaza todas las letras que tengan acentos por las que no tiene acentos. áà a, éàe, íài, óào, úàu. Va a ser llamada para sacarle los acentos al título ingresado por el usuario y para sacarle los acentos al título del libro que se encuentra almacenado en la lista. Su complejidad es O(n) siendo n la longitud de la palabra con acentos.
- Compararasterisco: comprara carácter a carácter el título ingresado por el usuario con el título del libro. Su complejidad es O(n) siendo n la longitud del título del libro.
- Compararsigno: es muy similar al compararasterisco. Su complejidad es O(n) siendo n la longitud de la palabra a comparar.

```
void Libreria::existe_libro (Lista<Libro> & libros, string titulo){

    titulo.erase(std::remove(titulo.begin(), titulo.end(),' '), titulo.end());

    sinacentos(titulo);

    transform(titulo.begin(), titulo.end(), titulo.begin(), ::tolower);

    libros.cursor_principio();
```

```
    for (unsigned int i=0; i < libros.longitud_lista() ; i++){
        Libro l = libros.recuperar_elem();
        string word =l.devolverTitulo();
        word.erase(std::remove(word.begin(), word.end(),' '), word.end());
        transform(word.begin(), word.end(), word.begin(), ::tolower);
        sinacentos(word);
        if ((titulo==word)||(compararAsterisco(titulo, word))||(compararSigno(titulo, word))){
            cout<<endl<< "Resultado/s: "<<endl<<endl;
            libros.mostrar();
        }
        libros.avanzar_cursor();

    }
}
```

En este servicio en específico usamos una librería propia de C++ *#include <algorithm>* para poder transformar un string que tenga mayúsculas en todas minúsculas. Tanbien la usamos para borrar espacios que puedan llegar a tener.

Complejidad: O(n^2) con n siendo la longitud de la lista de libros.

4.3 Servicio 2: Listar libros en un rango de fechas

El segundo servicio que se nos pide crear, es una búsqueda de libros en un rango de fechas. El usuario tiene que ingresar dos años, el año de inicio del rango y el año del fin del rango.

Para este servicio no creamos ninguna función auxiliar.

Funcion listar_por_fecha:

```
void listar_por_fecha(Lista<Libro> & libros, string fecha_usuario_1, string fecha_usuario_2 ){


    libros.cursor_principio();

    for (unsigned i=0; i< libros.longitud_lista(); i++){

    Libro l = libros.recuperar_elem();

    if (l.devolverFechaPublicacion() >= fecha_usuario_1){

            if (l.devolverFechaPublicacion() <= fecha_usuario_2){

            cout<<endl;

               libros.mostrar();

               cout<<endl;

            }

    }

    libros.avanzar_cursor();

    }

    cout << endl <<endl;

    cout << "Si no se muestra ningun libro es porque no existe ninguno con esa fecha." << endl << endl;

    cout << endl;

}
```

Complejidad: O(n) siendo n la longitud de la lista de libros que se tiene que recorrer.

4.4 Servicio 3: Libros económicos.

El tercer y último servicio pide listar entre 1(uno) y 10 (diez) libros más económicos que están disponibles en un idioma determinado, que va a ser ingresado por el usuario.

En este servicio en específico tuvimos varias dificultades, ya que sin duplicar en memoria los Libros no sabíamos cómo resolverlo.

Para resolverlo pensamos en almacenar en un arreglo de strings los isbn de los libros que tengan el mismo idioma que el que fue ingresado por el usuario. Usando este arreglo compararemos los precios de los libros que tengan ese isbn ordenándolos. De esta manera cuando busquemos en la lista de libros (siempre comparando los isbn del arreglo con el de la lista de libros) mostramos por pantalla (la cantidad ingresada por el usuario) los que tengan el mismo isbn..

Utilizamos varias funciones a la hora de resolver el servicio. Estos consisten en:

- cant_libros: Nos devuelve la cantidad de libros de un idioma en específico que hay cargados en la plataforma. Su complejidad es O(n^2) siendo n la longitud de la lista de libros y la longitud de la lista de idiomas de cada libro.
- mismo_idioma: Es una función booleana que devuelve true(verdadero) si encuentra un libro que esté disponible en un idioma en específico. Su complejidad es O(n) siendo n la longitud de la lista de idiomas disponibles de un libro.
- busqueda_precio: recorre la lista de libros y dado los isbn que están almacenados en un arreglo, devuelve el precio del encontrado. Su complejidad es de O(n) siendo n la longitud de la lista de libros.
- ordenar_arreglo: ordena el arreglo de isbn dependiendo del precio de cada libro que es representado por este isbn. Su complejidad es O(n^2) siendo n cant_libros (tam).
- guardar: guarda en el arreglo de strings los isbn de los libros que están disponibles en el mismo idioma que el ingresado por el usuario. Su complejidad es de O(n^2) siendo n la longitud de la lista de libros y de idiomas. Esto se debe a que en la función se llama a mismo_idioma dentro del for.

La función principal es mostrar_economicos:

```
void Libreria::mostrar_economicos (Lista<Libro> libros, string idioma, int cant_usuario){
    string nro_libros[100];
    guardar(libros,idioma, nro_libros);
    ordenar_arreglo(libros,idioma,nro_libros);
    unsigned int j=0;
    while (j<cant_usuario){
        libros.cursor_principio();
        for (unsigned int k=1; k<=libros.longitud_lista();k++){
            Libro i = libros.recuperar_elem();
            if (mismo_idioma(libros, idioma, i)){
                if (i.devolverIsbn() == nro_libros[j]){
                libros.mostrar();
                }
            }
            libros.avanzar_cursor();
        }
        j++;
    }
}
```

La complejidad de la función es O(n^3). Se podría mejorar para que la complejidad temporal sea menor.

La complejidad de esta función afecta al programa ya que es la tiene mayor complejidad temporal. Una solución que pensamos fue la de implementar un puntero que ordene la lista según el precio de cada libro. De esta manera solo se emplearía una función que cada vez que encuentre un libro con el mismo idioma que el ingresado por el usuario se mostraría por pantalla. Esto se podría hacer recorriendo la lista hasta que se mostraron todos los libros deseados.

5. Conclusión

Como conclusión, vemos que el código se podría mejorar mas al implementar otro tipo de estructuras como una lista doblemente enlazada, mas punteros o incluso un árbol. Si la estructura de almacenamiento fuera un árbol, esta seria muchísimo mas eficiente ya que no se recorrería innecesariamente, salvando tiempo y complejidad.

6. BIBLIOGRAFIA

http://www.cplusplus.com/reference/ostream/ostream/operator%3C%3C/

https://moodle.exa.unicen.edu.ar/pluginfile.php/34995/mod_resource/content/2/apunteNEREUS2020.pdf

https://sites.google.com/site/programacioniiuno/mi-cursada---turno-noche/bitacoras/30042013-clase4-tdalista

https://sites.google.com/site/programacioniiuno/temario/unidad-3---estructuras-de-datos-comunes-y-colecciones/la-estructura-lista

https://www.programiz.com/cpp-programming/templates

https://en.wikipedia.org/wiki/Template_metaprogramming

https://stackoverflow.com/questions/10688831/fastest-way-to-capitalize-words/31002777

https://www.geeksforgeeks.org/conversion-whole-string-uppercase-lowercase-using-stl-c/