

Trabajo Practico Especial

6 de Noviembre 2020

Grupo 25:

Fernandez Soler Marcos Agustin.

Carretto Leonardo Agustin.

Consigna

Objetivo:

Una plataforma eCommerce de Libros cuenta con una extensa base de datos de libros y se nos pide implementar un sistema de consulta para los clientes. Este sistema deberá llevar un registro de los libros con los que cuenta el sitio, y proveer distintos servicios de búsqueda que respondan a las consultas realizadas por los usuarios.

Servicios Requeridos:

La plataforma debe proveer un conjunto de servicios que deberán ser resueltos de la manera más eficiente posible. Los servicios de búsqueda que se deben proveer son:

1. Verificar si existe un título dado dentro de la colección de libros del sitio. Permitiendo usar comodines ("*" para una palabra y "?" para un carácter) y descartando acentos en la búsqueda.
2. Listar todos los libros publicados en algún rango de fechas en particular solicitado por el cliente, presentando de cada uno su información general y los idiomas en los que se encuentra disponible.
3. Obtener los N ($1 \leq N \leq 10$) libros más económicos de un idioma determinado para poder ofertar a los clientes. La información de los libros deberá ser obtenida de un archivo de texto al momento de iniciar la aplicación. Una vez cargada esta información la misma no cambiará hasta que la aplicación no se ejecute nuevamente.

La aplicación se iniciará al comienzo del día y se detendrá al finalizar la jornada; por lo tanto, seguramente se deban resolver un número considerable de veces cada uno de los servicios establecidos como necesarios. Se sabe que la plataforma cuenta con una colección muy grande de libros que se debe tener en cuenta al momento de pensar los servicios para hacerlos eficientes

RESUMEN

En este trabajo se presenta la Especificación e implementación en C++ de los TDA (Tipos de Datos Abstractos) y el “Main” necesarios para la resolución del enunciado propuesto por la cátedra como Trabajo Práctico Especial 2020 2do Cuatrimestre.

Desarrollo del Trabajo:

Estructura implementada:

Para el desarrollo de este trabajo decidimos implementar una lista compuesta de “nodos”, estos son structs que contienen un puntero a un tipo libro y un puntero a un tipo nodo.

TDAs Implementados:

Libro:

Es el Tipo de dato que almacena la información de cada libro del archivo de texto, este esta compuesto por el ISBN, Nombre, Autor, Año de publicación, País de origen, Precio e idiomas disponibles.

Este TDA cuenta con los Métodos :

libro(Constructor): Este método recibe 5 strings, un float y un int para crear una instancia del tipo libro.

imprimirLibro(obs): Este método se encarga de mostrar en consola todos los atributos del libro que se le pase por parámetro.

Getters(obs): permiten el acceso a los atributos del libro desde el public.

Operator = (mod): Definición del operador de asignacion para permitir la manipulación de la clase.

Libreria:

Este tipo de dato es el encargado de almacenar todas las instancias de libro que se creen para poder acceder a ellos. Inicialmente solo cuenta con un puntero nulo.

Este TDA cuenta con los Métodos :

Libreria(constructor): Inicializa un puntero como nullptr.

agregarLista(mod):Este método se encarga de crear un nuevo nodo y pasarlo al método agregarNodo.

agregarNodo(mod): Recibe por parámetro un puntero a nodo y recorre la lista para insertarlo donde se deba según el precio del libro que contiene.

imprimirEconomicos(obs): Este método recorre la lista comprobando si entre los idiomas de cada libro se encuentra el que se le haya pasado por parámetro, ejecutando imprimirLibro cada vez que encuentre uno y haya encontrado menos de 10.

imprimirEntreFechas(obs):Este método recibe dos enteros como parámetros y recorre la lista buscando todos los nodos que contengan un libro cuya fecha esté entre ambas cotas, imprimiendo todos los que cumplan este requisito.

buscarPorTitulo(obs):Este método recibe un string como entrada y recorre la lista ejecutando coincidencia(valor bool), en caso de ser True, buscarPorTitulo también devuelve True.

coincidencia(obs): Convierte un string de entrada y el nombre del libro que se le mande a strings de C(c_str) cuya estructura es un Array de Char, y luego copia ambos a arrays de char de C++, una vez hecho esto le quita los acentos a todas las vocales, haciéndolas pasar por la función removeAccent y luego compara ambos arreglos teniendo en cuenta que esta habilitado el uso de comodines en al búsqueda, tratando de forma especial el caracter * (Comodín de una palabra) y ? (Comodín de una letra).

Analisis de los TDA

Especificacion NEREUS

Class Libro

IMPORTS String, Int, Float,

TYPE Libro

Basic Constructors libro;

EFFECTIVE

OPERATIONS:

libro: string*string*string*int*string*float*string -> Libro; //constructor

getIdioma: libro -> string; //obs

getIsbn: libro->string; //obs

getPais: libro->string; //obs

getPrice:libro->float; //obs

getDate: libro->int; //obs

getAuthor:libro->string; //obs

getTitle:libro->string; //obs

imprimirLibro:libro->int; //obs

CLASS Libreria

IMPORTS Libro, Int, Float, String, nodo.

BASIC CONSTRUCTOR Libreria, agregarLista.

EFFECTIVE

TYPE Libreria

OPERATIONS

Libreria: -> Libreria; //constructora

agregarLista: Libreria*libro->Libreria; //constructor

insertarNodo: Libreria*nodo->Libreria; //mod

agregarNodo: Libreria*nodo->Libreria; //mod

coincidencia:libro*string->bool; //obs

idiomaDisponible:string*nodo->bool; //obs

imprimirEconomicos: string->bool; //obs

imprimirEntreFechas:int*int->; //obs

buscarPorTitulo:string->bool; //obs

Analisis de Complejidad Temporal:

En este análisis se empleara pseudocódigo y complejidades descriptivas (solo se muestra su comportamiento, no la función a aplicar a la entrada)

buscarPorTitulo: Consta de un while donde en cada iteración pregunta por coincidencia, este le pasa por parámetro un libro(del cual extrae un string nombre) y un string.

Cantidad de iteraciones: n (Longitud de lista)

Longitud de string = m

$T(\text{strcpy}) \in O(m)$ m = Longitud de la cadena de entrada.

ProcRemoveAccent: loop que ejecuta removeAccent (tiempo constante) m cantidad de veces.

$T(\text{ProcRemoveAccent}) \in O(m)$ m = Longitud de la cadena de entrada

ProcesoMatch= Nombre que le asignaremos para el calculo de eficiencia al conjunto de operaciones que determinan el valor de coincidencia

$T(\text{ProcesoMatch}) \in O(m)$ m = Longitud de la cadena más larga entre sus 2 entradas.

$T(\text{coincidencia}) \leq T(\text{strcpy}(m)) + T(\text{strcpy}(k)) + T(\text{removeAccents}(k)) + T(\text{procesoMatch}(\max\{m;k\}))$

Donde m y k son ambos strings de entrada.

Entonces $T(\text{coincidencia}) \in O(\max\{m;k\})$

Entonces $T(\text{buscarPorTitulo}) \leq n * T(\text{coincidencia}) \rightarrow T(\text{buscarPorTitulo}) \in O(n * g)$
donde n = longitud de lista y g =longitud del string mas largo de toda la lista.

ImprimirEntreFechas: Este método recibe dos ints de entrada y recorre la lista buscando valores que se encuentren entre esas cotas. Al tener que recorrer toda la lista para

encontrar las coincidencias tiene una complejidad del tipo $T(\text{imprimirEntreFechas}) \in O(n)$ donde $n = \text{longitud de lista}$.

idiomaDisponible: Tiene todos constantes y un find, por lo tanto como $T(\text{find}) \in O(n) \rightarrow T(\text{idiomaDisponible}) \in O(n)$. Donde $n = \text{longitud del string de entrada}$. \in

imprimirEconomicos: Recorre la lista ejecutando idiomaDisponible en busca de coincidencias de idioma. Al iterar por la lista podemos ver que se tiene que $T(\text{imprimirEconomicos}) = m * T(\text{idiomaDisponible})$, donde $m = \text{longitud de lista}$.

Entonces $T(\text{imprimirEconomicos}) \in O(n*m)$

insertarNodo: Tiene tiempo constante.

agregarNodo:

este método recorre la lista y hace un llamado a insertarNodo, entonces $T(\text{agregarNodo}) \leq n * T(\text{insertarNodo}) \rightarrow T(\text{agregarNodo}) \in O(n)$ Donde $n = \text{Longitud de lista}$.

AgregarLista:

Este método cuenta con operaciones elementales y un llamado a agregarNodo, entonces $T(\text{agregarLista}) \leq C + T(\text{agregarNodo})$, como $T(\text{agregarNodo}) \in O(n)$, entonces $T(\text{agregarLista}) \in O(n)$ Donde $n = \text{Longitud de lista}$.

~Libreria: Recorre toda la lista y al volver va eliminando todos los elementos, por lo tanto $T(\sim \text{Libreria}) \in O(n)$ Donde $n = \text{Longitud de lista}$.

Libreria: Pone el puntero en null, tiempo constante.

Libro:

convertirAString: Concatena los elementos de un arreglo, entonces $T(\text{convertirAString}) \in O(m)$ Donde $m = \text{tamaño del arreglo}$.

Además de las operaciones elementales, este constructor tiene un `ProcRemoveAccent` ($\in O(m)$) y un `convertirAString` ($\in O(m)$), por lo tanto $T(\text{Libro}) \in O(m)$ Donde $m = \text{longitud del string "idiomas"}$.

Libro &Operator = Tiempo constante.

imprimirLibro = Tiempo constante.

getTitle=Tiempo constante.

getDate=Tiempo constante.

getPrice=Tiempo constante.

getPais=Tiempo constante.

getAuthor=Tiempo constante.

getIsbn=Tiempo constante.

getIdiomas=Tiempo constante.

Comentarios

*Como la estrategia empleada no requería el uso del mismo contenedor para varios tipos de datos no se implementó un contenedor genérico.

*No se hizo uso de referencias constantes por el hecho de estar usando punteros hacia objetos en lugar de objetos.

*Todas las complejidades de las funciones no implementadas por nosotros fueron consultadas en su respectivo apartado en la página de C++.

*Optamos por implementar todo el código dentro del .h en lugar de un .h y un .cpp por preferencias personales.

*Hicimos un cambio al Main proporcionado por la cátedra, donde se leía “fecha de publicación” como un string, lo cambiamos a un int mediante el uso del comando Atoi y c_str.

*Modificamos el archivo .csv para facilitar la prueba del programa, ya que este tenía espacios en blancos al final de cada nombre. Este cambio no afecta la funcionalidad del programa, sino que hace más cómodo su uso.

*En lugar de completar y utilizar el código brindado por la catedra para cortar el string “idiomas” implementamos una función que busca dentro del string completo. Este cambio significa una mejora tanto en tiempo de procesamiento como en memoria utilizada.

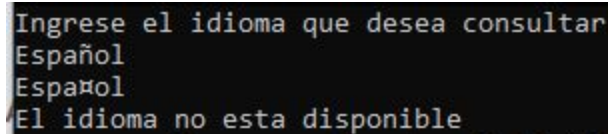
*Optamos por un orden ascendente en precio para la lista, beneficiando el tercer servicio (Imprimir los 10 más económicos de un idioma) , sin perjudicar los otros servicios.

*La técnica empleada en la búsqueda por comodines consiste en convertir el string de entrada a un string de C (el cual es considerado un array de char), luego pasar este a un array de C++. Una vez hecho esto usamos un algoritmo de comparación entre arreglos tratando especialmente los símbolos de comodín.

*Usamos recorridos iterativos en vez de recursivos (Excepto para la creación de la lista) por la optimización espacial que representa.

*Ya que el comando "cin" reconoce el espacio como fin de la cadena, optamos por el uso de getline(cin,string) donde string es la variable que almacenará la entrada.

*En el constructor de libro removimos los acentos de los idiomas(con el mismo mecanismo que en el servicio 1) por una interacción errónea entre el comando getline(cin,string) y las letras con acento, donde al introducirlas por consola se tomaban como un caracter erróneo (en un caso devuelve un acento, en otro caso borraba la letra y en otro devolvía el signo ϕ) cosa que no ocurría si se llamaba al método pasándole el string desde el código (por ejemplo "L.imprimirEconomicos("Nórdico antiguo")") Dicho problema tambien ocurre con la letra "ñ" pero no pudimos contemplarlo.



```
Ingrese el idioma que desea consultar
Español
Español
El idioma no esta disponible
```

La imagen adjunta muestra el input y el output al realizar una búsqueda con la letra "ñ"