

Algoritmos y Estructuras de Datos I

**PROYECTO INDIVIDUAL – PARTE II**

Francisco Piria

# Contenido

[Contenido 2](#_Toc514632772)

[Introducción 3](#_Toc514632773)

[Problema planteado 3](#_Toc514632774)

[Análisis de alternativas 3](#_Toc514632775)

[Primera alternativa 3](#_Toc514632776)

[Segunda alternativa 6](#_Toc514632777)

[Algoritmos 8](#_Toc514632778)

[Primera alternativa 8](#_Toc514632779)

[Segunda alternativa 15](#_Toc514632780)

[Comparación de alternativas 21](#_Toc514632781)

[Selección y justificación de alternativa a implementar 22](#_Toc514632782)

[Conclusiones 23](#_Toc514632783)

[Guía de usuario 23](#_Toc514632784)

# Introducción

El trabajo realizado consiste de una aplicación Java de consola para la empresa UcuBooks, que permite gestionar una biblioteca virtual de libros. El principal uso de esta aplicación es permitirle a los clientes realizar búsquedas de libros por diferentes criterios, entre otros.

## Problema planteado

Las tareas que debe poder realizar la aplicación incluyen:

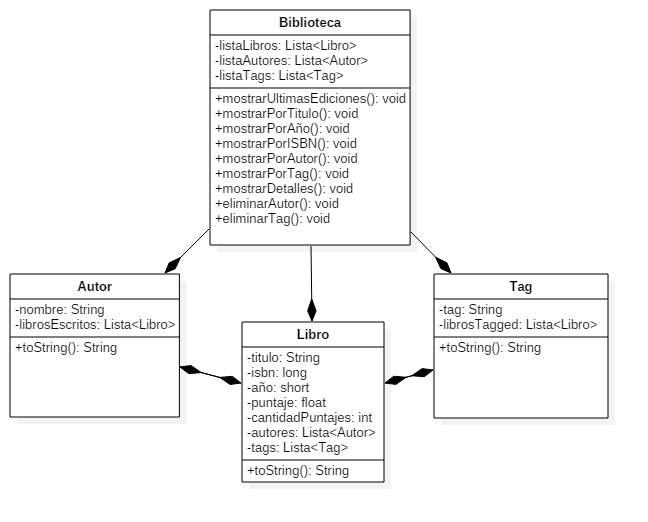
* Permitir hacer búsquedas de libros por nombre, autor, tag, año o ISBN y mostrar en pantalla los resultados
* Mostrar en pantalla los detalles de un libro en particular
* Mostrar las últimas ediciones a partir de un cierto año
* Eliminar un autor y todos sus libros
* Eliminar un tag y removerlo de todos sus libros

Todas estas funcionalidades se deben poder realizar a través de líneas de comando.

# Análisis de alternativas

A continuación, se describirán dos alternativas para solucionar las tareas que exige UcuBooks, las cuales implementan el tipo de datos abstracto Lista y Árbol Binario.

### Primera alternativa

La primera alternativa al problema implementa solamente el TDA Lista, y tiene esta estructura:

En esta alternativa, se tienen tres clases que representan las entidades libro, autor y tag. Cada autor contiene una lista con los libros que escribió, y cada tag contiene una lista de los libros que llevan ese tag (solo referencias a los libros). A su vez, cada libro contiene una lista de los autores que lo escribieron y una lista de sus tags. La biblioteca entonces, contiene tres listas: una para libros, otra para autores y otra para tags.

A continuación, se ven los costos de memoria de esta alternativa, comenzando por el de cada libro en la biblioteca:

|  |  |
| --- | --- |
| **Lista de libros** | **Espacio** |
| Titulo - String | 4 B |
| ISBN - long | 8 B |
| Año - short | 2 B |
| Puntaje - float | 4 B |
| Cantidad de puntajes - int | 4 B |
| Lista de referencias a autores[[1]](#footnote-1) | 20B nodos + 20B referencias a siguientes |
| Lista de referencias a tags[[2]](#footnote-2) | 600B nodos + 600B referencias a siguientes |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **1266 B** |

Ahora se van a analizar los costos de cada tag y cada autor:

|  |  |
| --- | --- |
| **Lista de autores** | **Espacio** |
| Nombre - String | 4 B |
| Lista de referencias a libros[[3]](#footnote-3) | 40B nodos + 40B referencias a siguientes |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **88 B** |

|  |  |
| --- | --- |
| **Lista de tags** | **Espacio** |
| Tag - String | 4 B |
| Lista de referencias a libros[[4]](#footnote-4) | 40B nodos + 40B referencias a siguientes |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **88 B** |

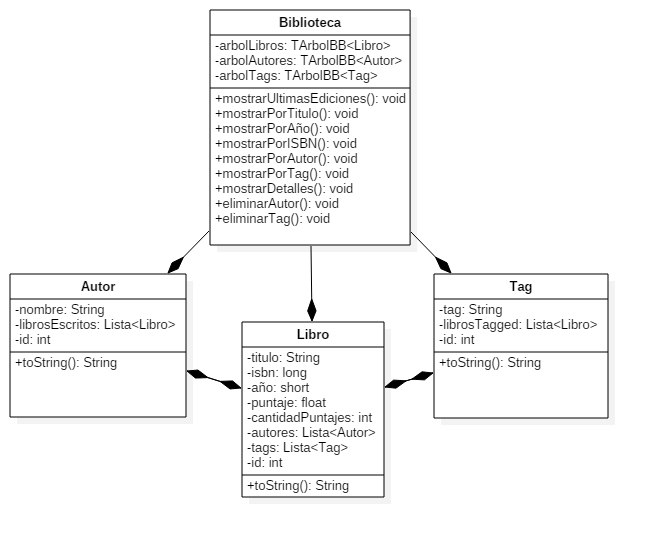
Tambien vamos a calcular un estimado del total de memoria necesaria

|  |  |
| --- | --- |
|  | **Espacio** |
| Lista de libros – aprox. 5800 elementos | 7342800 B |
| Lista de autores – aprox. 4800 elementos | 422400 B |
| Lista de tags – aprox. 34000 elementos | 2992000 B |
| **Total** | **10757200 B = 10,8 MB** |

Vemos que la memoria requerida crece rápidamente, debido a la gran cantidad de referencias entre todos los objetos que maneja el programa. A medida que crece la cantidad de elementos para manejar, la cantidad de asociaciones entre ellos crece mucho mas rápido, y eso es lo que causa este gran consumo de memoria.

### Segunda alternativa

La segunda alternativa al problema implementa tanto al TDA Lista como al TDA Árbol Binario. Tiene esta estructura:



Esta alternativa es muy similar a la anterior, la única diferencia siendo que la clase biblioteca ahora utiliza arboles binarios para almacenar los libros, autores y tags. Estos árboles binarios son AVL, de forma que permiten que las operaciones de búsqueda se realicen más rápidamente.

Todas las operaciones que realiza la clase Biblioteca son en realidad delegadas a los árboles que contienen los elementos. Mas adelante se muestra un diagrama de clases mas detallado con respecto a eso.

A continuación, se ven los costos de memoria de esta alternativa, comenzando por el de cada libro en la biblioteca:

|  |  |
| --- | --- |
| **Lista de libros** | **Espacio** |
| Id - int | 4 B |
| Titulo - String | 4 B |
| ISBN - long | 8 B |
| Año - short | 2 B |
| Puntaje - float | 4 B |
| Cantidad de puntajes - int | 4 B |
| Lista de referencias a autores[[5]](#footnote-5) | 20B nodos + 20B referencias a siguientes |
| Lista de referencias a tags[[6]](#footnote-6) | 600B nodos + 600B referencias a siguientes |
| Referencias en el nodo a sus dos hijos | 8 B |
| **Total** | **1274 B** |

Ahora se van a analizar los costos de cada tag y cada autor:

|  |  |
| --- | --- |
| **Lista de autores** | **Espacio** |
| Id - int | 4 B |
| Nombre - String | 4 B |
| Lista de referencias a libros[[7]](#footnote-7) | 40B nodos + 40B referencias a siguientes |
| Referencias en el nodo a sus dos hijos | 8 B |
| **Total** | **96 B** |

|  |  |
| --- | --- |
| **Lista de tags** | **Espacio** |
| Id - int | 4 B |
| Tag - String | 4 B |
| Lista de referencias a libros[[8]](#footnote-8) | 40B nodos + 40B referencias a siguientes |
| Referencias en el nodo a sus dos hijos | 8 B |
| **Total** | **96 B** |

Igual que para el caso anterior, vamos a calcular un estimado del total de memoria necesaria

|  |  |
| --- | --- |
|  | **Espacio** |
| Lista de libros – aprox. 5800 elementos | 7389200 B |
| Lista de autores – aprox. 4800 elementos | 460800 B |
| Lista de tags – aprox. 34000 elementos | 3264000 B |
| **Total** | **11114000 B = 11,1 MB** |

En este caso la memoria requerida también crece rápidamente, por las mismas razones que en el caso anterior. El costo es un poco mas alto por el uso de árboles en lugar de listas.

## Algoritmos

Antes de comenzar las descripciones en seudocódigo y análisis de ordenes de tiempo de ejecución para las funcionalidades requeridas, sería adecuado explicar brevemente las operaciones de los TDAs involucrados, dado que son utilizados por ambas alternativas.

Las operaciones más importantes que realiza el TDA Lista (que se estarán utilizando constantemente en los otros algoritmos) son insertar, buscar, eliminar, quitar y buscarLista.

* La operación **insertar** agrega un nuevo elemento al final de la lista, y su orden de tiempo de ejecución es O(n)
* **buscar** devuelve un nodo de la lista cuya etiqueta coincide con la clave pasada por parámetro. Su orden también es O(n)
* **eliminar** remueve y anula un nodo de la lista, cuya etiqueta coincide con la clave pasada por parámetro. O(n)
* **quitar** remueve y devuelve un nodo de la lista, cuya etiqueta coincide con la clave pasada por parámetro. O(n)
* **buscarLista** funciona igual que **buscar**, solo que devuelve una lista de los nodos que coinciden con la clave dada. O(n)

Por otro lado, tenemos también las operaciones del TDA Árbol Binario. Los arboles utilizados son AVL, esto quiere decir que están balanceados y cada vez que se realiza una modificación al árbol que lo desbalancea (por ejemplo, insertar o eliminar), éste automáticamente realiza las rotaciones necesarias para restaurar el balance.

* La operación **insertar** agrega un nuevo elemento al final de la lista, y su orden de tiempo de ejecución es O(log(n)) gracias a que el árbol esté balanceado.
* **buscar** devuelve un nodo de la lista cuya etiqueta coincide con la clave pasada por parámetro. Su orden también es O(log(n)).
* **eliminar** remueve y anula un nodo de la lista, cuya etiqueta coincide con la clave pasada por parámetro. O(log(n)).
* Los recorridos en **preorden**, **postorden** e **inorden** nos muestran todas las claves del árbol. O(n).
* Todas las operaciones de rebalanceo del árbol son O(1).

Ahora si pasamos a los métodos de cada alternativa

### Primera alternativa

#### mostrarUltimasEdiciones

##### Lenguaje Natural

Recorre la lista de libros, imprimiendo los libros cuyo año es mayor o igual al año dado

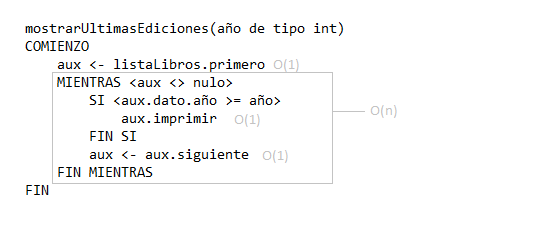
##### Precondiciones

* La lista contiene una n cantidad de nodos, cada nodo contiene un libro que a su vez contiene un campo “año”

##### Postcondiciones

* La lista no se modificó
* Se imprimió en pantalla los libros que coinciden con el criterio, en caso de haberlos

##### Seudocódigo



El orden del tiempo de ejecución de este algoritmo resulta ser de O(n).

#### mostrarPorTitulo

##### Lenguaje Natural

Recorre la lista de libros, imprimiendo los libros cuyo título es igual al título dado

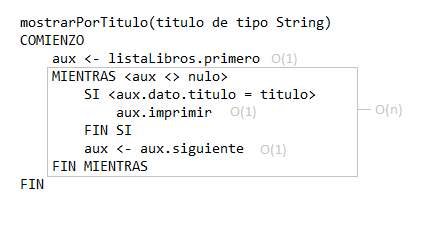
##### Precondiciones

* La lista contiene una n cantidad de nodos, cada nodo contiene un libro que a su vez contiene un campo “titulo”

##### Postcondiciones

* La lista no se modificó
* Se imprimió en pantalla los libros que coinciden con el criterio, en caso de haberlos

##### Seudocódigo



El orden de este algoritmo es entonces de O(n).

#### mostrarPorAño

##### Lenguaje Natural

Recorre la lista de libros, imprimiendo los libros cuyo año es igual al año dado

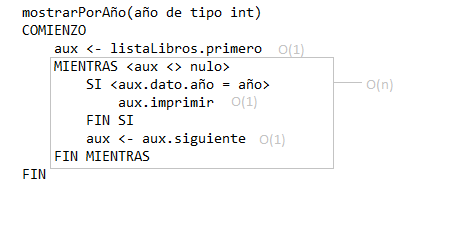
##### Precondiciones

* La lista contiene una n cantidad de nodos, cada nodo contiene un libro que a su vez contiene un campo “año”

##### Postcondiciones

* La lista no se modificó
* Se imprimió en pantalla los libros que coinciden con el criterio, en caso de haberlos

##### Seudocódigo



El orden de este algoritmo es entonces de O(n).

#### mostrarPorISBN

##### Lenguaje Natural

Busca en la lista de libros aquel que tiene el ISBN dado, y luego lo imprime.

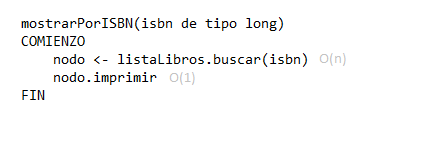
##### Precondiciones

* La lista contiene una n cantidad de nodos, cada nodo contiene un libro y su etiqueta es el ISBN del libro.

##### Postcondiciones

* La lista no se modificó
* Se imprimió en pantalla el libro que coincide con el criterio, en caso de haberlo

##### Seudocódigo



El orden de este algoritmo es entonces de O(n).

#### mostrarPorAutor

##### Lenguaje Natural

Busca en la lista de autores aquel que tiene el nombre dado, y luego imprime la lista de libros escritos por él.

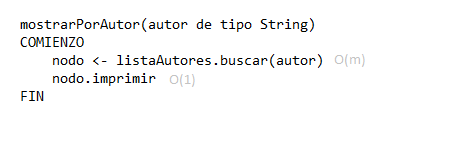
##### Precondiciones

* La lista de autores contiene una m cantidad de nodos, cada nodo contiene un autor que a su vez contiene una lista con las referencias a los libros que escribió (y la etiqueta del nodo es el nombre del autor mismo).
* El autor que se busca existe en la biblioteca.

##### Postcondiciones

* Ninguna de las listas se modificó
* Se imprimió en pantalla los libros que coinciden con el criterio, en caso de haberlos

##### Seudocódigo



El orden de este algoritmo es entonces de O(m).

#### mostrarPorTag

##### Lenguaje Natural

Busca en la lista de tags aquel que tiene el nombre dado, y luego imprime la lista de libros que llevan ese tag.

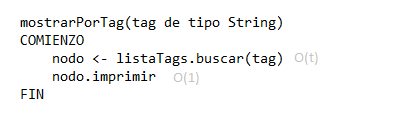
##### Precondiciones

* La lista de tags contiene una t cantidad de nodos, cada nodo contiene un tag que a su vez contiene una lista con las referencias a los libros que llevan ese tag (y la etiqueta del tag es el String tag mismo).
* El tag que se busca existe en la biblioteca.

##### Postcondiciones

* Ninguna de las listas se modificó
* Se imprimió en pantalla los libros que coinciden con el criterio, en caso de haberlos

##### Seudocódigo



El orden de este algoritmo es entonces de O(t).

#### mostrarDetalles

##### Lenguaje Natural

Busca en la lista de libros aquel que tiene el ISBN dado, y luego imprime sus detalles (todos sus datos con autores y tags).

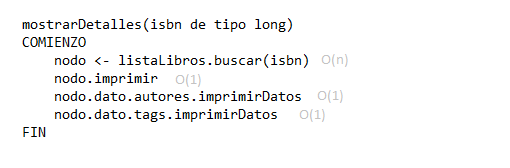
##### Precondiciones

* La lista contiene una n cantidad de nodos, cada nodo contiene un libro y su etiqueta es el ISBN del libro.

##### Postcondiciones

* La lista no se modificó
* Se imprimió en pantalla el libro que coincide con el criterio, en caso de haberlo

##### Seudocódigo



El orden de este algoritmo es entonces de O(n).

#### eliminarAutor

##### Lenguaje Natural

Busca en la lista de autores el autor dado y luego recorre su lista de libros escritos. Para cada uno de los libros que escribió, vamos a la lista de libros y lo eliminamos. Finalmente eliminamos al autor de la lista de autores.

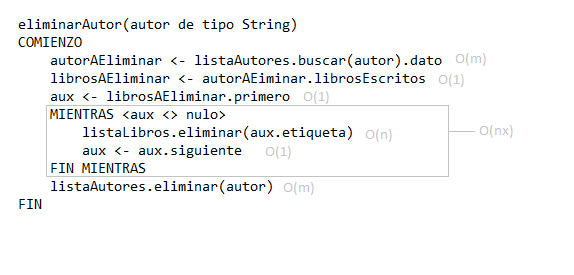
##### Precondiciones

* La lista de libros contiene una n cantidad de nodos, cada nodo contiene un libro y su etiqueta es el ISBN del libro.
* La lista de autores contiene una m cantidad de nodos, cada nodo contiene un autor que a su vez contiene una lista con las referencias a los libros que escribió (y la etiqueta del nodo es el nombre del autor mismo).
* El autor a eliminar existe en la biblioteca.

##### Postcondiciones

* Se eliminó al autor dado por parámetro y la lista de autores ahora contiene un elemento menos.
* Se eliminaron todos los libros del autor en la lista de libros.

##### Seudocódigo



Se le llamo “x” a la cantidad de libros a ser eliminados. Dado que esta cantidad es muy pequeña en comparación a los tamaños de las listas que se manejan, se puede considerar al “mientras” como de orden O(n). Entonces por la regla de la suma el orden de todo el algoritmo es O(n) (Dado que n es mayor a m).

#### eliminarTag

##### Lenguaje Natural

Busca en la lista de tags el tag dado y luego recorre su lista de libros taggeados. Para cada uno de los libros, vamos a su lista de tags y eliminamos el tag. Finalmente eliminamos el tag de la lista de tags de la biblioteca.

##### Precondiciones

* La lista de libros contiene una n cantidad de nodos, cada nodo contiene un libro y su etiqueta es el ISBN del libro.
* La lista de tags contiene una t cantidad de nodos, cada nodo contiene un tag que a su vez contiene una lista con las referencias a los libros que llevan ese tag (y la etiqueta del nodo es el tag mismo).
* El tag a eliminar existe en la biblioteca.

##### Postcondiciones

* Se eliminó al tag dado por parámetro y la lista de tags ahora contiene un elemento menos.
* Se eliminó al tag de todos los libros que lo tenian.

##### Seudocódigo

##### 

Se le llamo “x” a la cantidad de tags que contiene cada libro, e “y” a la cantidad de libros que llevan el tag dado. Como estas cantidades son muy pequeñas en comparación a los tamaños de las listas que se manejan, se puede considerar al “mientras” como de orden O(1). Entonces por la regla de la suma el orden de todo el algoritmo es O(t).

### Segunda alternativa

Todos los métodos de búsqueda son realizados a nivel de árbol. Los dos métodos de eliminación son realizados a nivel de biblioteca.

#### mostrarUltimasEdiciones

##### Lenguaje Natural

Recorre el árbol de libros en inorden, insertando los libros cuyo año es mayor o igual al año dado en una lista enlazada que será devuelta.

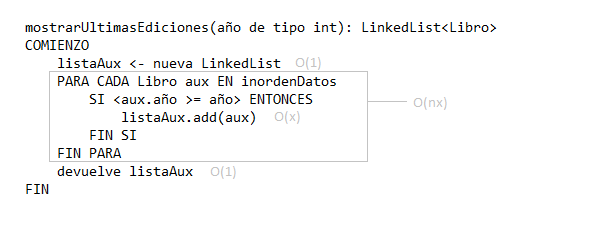
##### Precondiciones

* El árbol contiene una n cantidad de nodos, cada nodo contiene un libro que a su vez contiene un campo “año”

##### Postcondiciones

* El árbol no se modificó
* Se devolvieron los libros que coinciden con el criterio, en caso de haberlos

##### Seudocódigo



Se decidió utilizar una LinkedList para agrupar los libros que se devuelven, dado que es más fácil realizar iteraciones sobre ella que sobre el TDA Lista.

Se le llamo “x” a la cantidad de libros que coinciden con el criterio. Dado que esta cantidad es muy pequeña en comparación al tamaño del árbol de libros, se puede considerar al algoritmo como de orden O(n).

#### mostrarPorTitulo

##### Lenguaje Natural

Busca en el árbol de libros aquel cuyo título es igual al título dado, y lo devuelve.

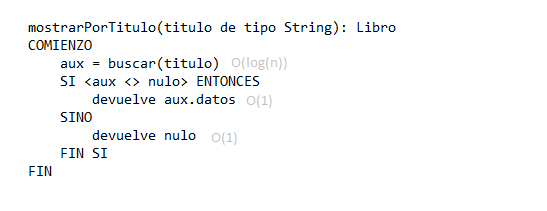
##### Precondiciones

* El árbol contiene una n cantidad de nodos, cada nodo contiene un libro y su etiqueta es el título del libro.

##### Postcondiciones

* El árbol no se modificó
* Se devolvieron los libros que coinciden con el criterio, en caso de haberlos

##### Seudocódigo



El orden de este algoritmo es entonces de O(log(n)).

#### mostrarPorAño

##### Lenguaje Natural

Recorre el árbol de libros en inorden, insertando los libros cuyo año es igual al año dado en una lista enlazada que será devuelta.

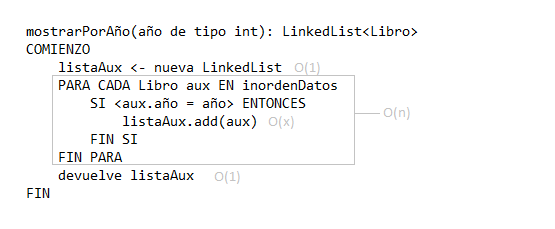
##### Precondiciones

* El árbol contiene una n cantidad de nodos, cada nodo contiene un libro que a su vez contiene un campo “año”

##### Postcondiciones

* El árbol no se modificó
* Se devolvieron los libros que coinciden con el criterio, en caso de haberlos

##### Seudocódigo



Al igual que antes, se utilizó una LinkedList para almacenar los libros devueltos, y se le llamo “x” a esa cantidad. El orden de este algoritmo es entonces de O(n).

#### mostrarPorISBN

##### Lenguaje Natural

Recorre el árbol de libros en inorden, buscando aquel que tiene el ISBN dado, y luego lo devuelve.

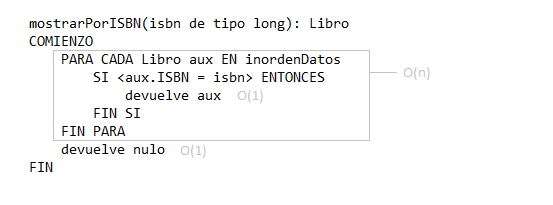
##### Precondiciones

* El árbol contiene una n cantidad de nodos, cada nodo contiene un libro que a su vez contiene un campo “isbn”.

##### Postcondiciones

* El árbol no se modificó
* Se devolvió el libro que coincide con el criterio, en caso de haberlo

##### Seudocódigo



El orden de este algoritmo es entonces de O(n).

#### mostrarPorAutor

##### Lenguaje Natural

Busca en el árbol de autores aquel que tiene el nombre dado, y luego devuelve la lista de libros escritos por él.

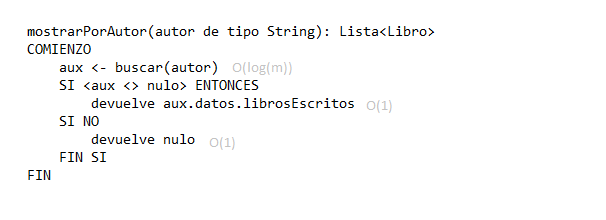
##### Precondiciones

* El árbol de autores contiene una m cantidad de nodos, cada nodo contiene un autor que a su vez contiene una lista con las referencias a los libros que escribió (y la etiqueta del nodo es el nombre del autor mismo).
* El autor que se busca existe en la biblioteca.

##### Postcondiciones

* Ninguna de las listas ni el árbol se modificaron
* Se devolvieron los libros que coinciden con el criterio, en caso de haberlos

##### Seudocódigo



El orden de este algoritmo es entonces de O(log(m)).

#### mostrarPorTag

##### Lenguaje Natural

Busca en el árbol de tags aquel que tiene el nombre dado, y luego imprime la lista de libros que llevan ese tag.

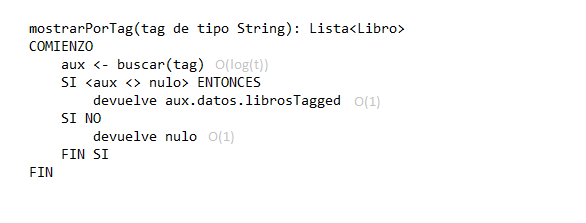
##### Precondiciones

* El árbol de tags contiene una t cantidad de nodos, cada nodo contiene un tag que a su vez contiene una lista con las referencias a los libros que llevan ese tag (y la etiqueta del tag es el String tag mismo).
* El tag que se busca existe en la biblioteca.

##### Postcondiciones

* Ninguna de las listas ni el árbol se modificaron
* Se devolvieron los libros que coinciden con el criterio, en caso de haberlos

##### Seudocódigo



El orden de este algoritmo es entonces de O(log(t)).

#### mostrarDetalles

##### Lenguaje Natural

Busca en la lista de libros aquel que tiene el título dado, y luego imprime sus detalles (todos sus datos con autores y tags).

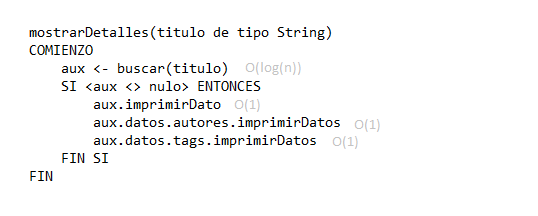
##### Precondiciones

* El árbol contiene una n cantidad de nodos, cada nodo contiene un libro y su etiqueta es el título del libro.

##### Postcondiciones

* El árbol no se modificó
* Se imprimió en pantalla el libro que coincide con el criterio, en caso de haberlo

##### Seudocódigo



El orden de este algoritmo es entonces de O(log(n)).

#### eliminarAutor

##### Lenguaje Natural

Busca en el árbol de autores el autor dado y luego recorre su lista de libros escritos. Para cada uno de los libros que escribió, vamos al árbol de libros y lo eliminamos. Finalmente eliminamos al autor del arbol de autores.

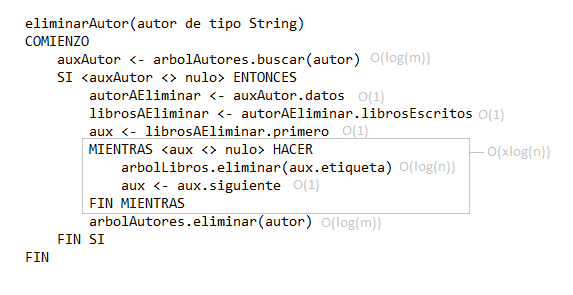
##### Precondiciones

* El árbol de libros contiene una n cantidad de nodos, cada nodo contiene un libro y su etiqueta es el título del libro.
* El árbol de autores contiene una m cantidad de nodos, cada nodo contiene un autor que a su vez contiene una lista con las referencias a los libros que escribió (y la etiqueta del nodo es el nombre del autor mismo).
* El autor a eliminar existe en la biblioteca.

##### Postcondiciones

* Se eliminó al autor dado por parámetro y el árbol de autores ahora contiene un elemento menos.
* Se eliminaron todos los libros del autor en el árbol de libros.

##### Seudocódigo



Se le llamo “x” a la cantidad de libros a ser eliminados. Dado que esta cantidad es muy pequeña en comparación a los tamaños de las listas que se manejan, se puede considerar al “mientras” como de orden O(log(n)). Entonces por la regla de la suma el orden de todo el algoritmo es O(log(n)) (Dado que n es mayor a m).

#### eliminarTag

##### Lenguaje Natural

Busca en el árbol de tags el tag dado y luego recorre su lista de libros taggeados. Para cada uno de los libros, lo buscamos en el árbol de libros y eliminamos el tag de su lista de tags. Finalmente eliminamos el tag del árbol de tags de la biblioteca. Si algún libro quedó sin tags, lo eliminamos del árbol de libros y lo agregamos a un árbol aparte.

##### Precondiciones

* El árbol de libros contiene una n cantidad de nodos, cada nodo contiene un libro y su etiqueta es el título del libro.
* El árbol de tags contiene una t cantidad de nodos, cada nodo contiene un tag que a su vez contiene una lista con las referencias a los libros que llevan ese tag (y la etiqueta del nodo es el tag mismo).
* El tag a eliminar existe en la biblioteca.

##### Postcondiciones

* Se eliminó al tag dado por parámetro y el árbol de tags ahora contiene un elemento menos.
* Se eliminó al tag de todos los libros que lo tenían.

##### Seudocódigo

##### 

Se le llamo “x” a la cantidad de tags que contiene cada libro, “z” a la cantidad de libros que llevan el tag dado, e “y” a la cantidad de libros en la biblioteca con no tienen tags. De nuevo, como estas cantidades son muy pequeñas en comparación a los tamaños de las listas que se manejan, se puede considerar al “mientras” como de orden O(log(n)). Entonces por la regla de la suma el orden de todo el algoritmo es O(log(t)) (dado que t es mayor que n).

### Comparación de alternativas

Llegó la hora de comparar las dos alternativas para ver cual soluciona mejor el problema. A modo de resumen, a continuación se presenta una tabla con los criterios que se tienen en cuenta:

|  |  |  |
| --- | --- | --- |
|  | **Ordenes de tiempo de ejecución** | |
| **Funcionalidades requeridas** | **Alternativa 1** | **Alternativa 2** |
| Buscar por título | O(n) | O(log(n)) |
| Buscar por año o ISBN | O(n) | O(n) |
| Buscar por tag | O(t) | O(log(t)) |
| Buscar por autor | O(m) | O(log(m)) |
| Buscar últimas ediciones | O(n) | O(n) |
| Mostrar detalles de un libro | O(n) | O(n) |
| Eliminar autor y sus libros | O(n) | O(log(n)) |
| Eliminar un tag | O(t) | O(log(t)) |
|  |  |  |
| Memoria requerida | 10,8 MB | 11,1 MB |

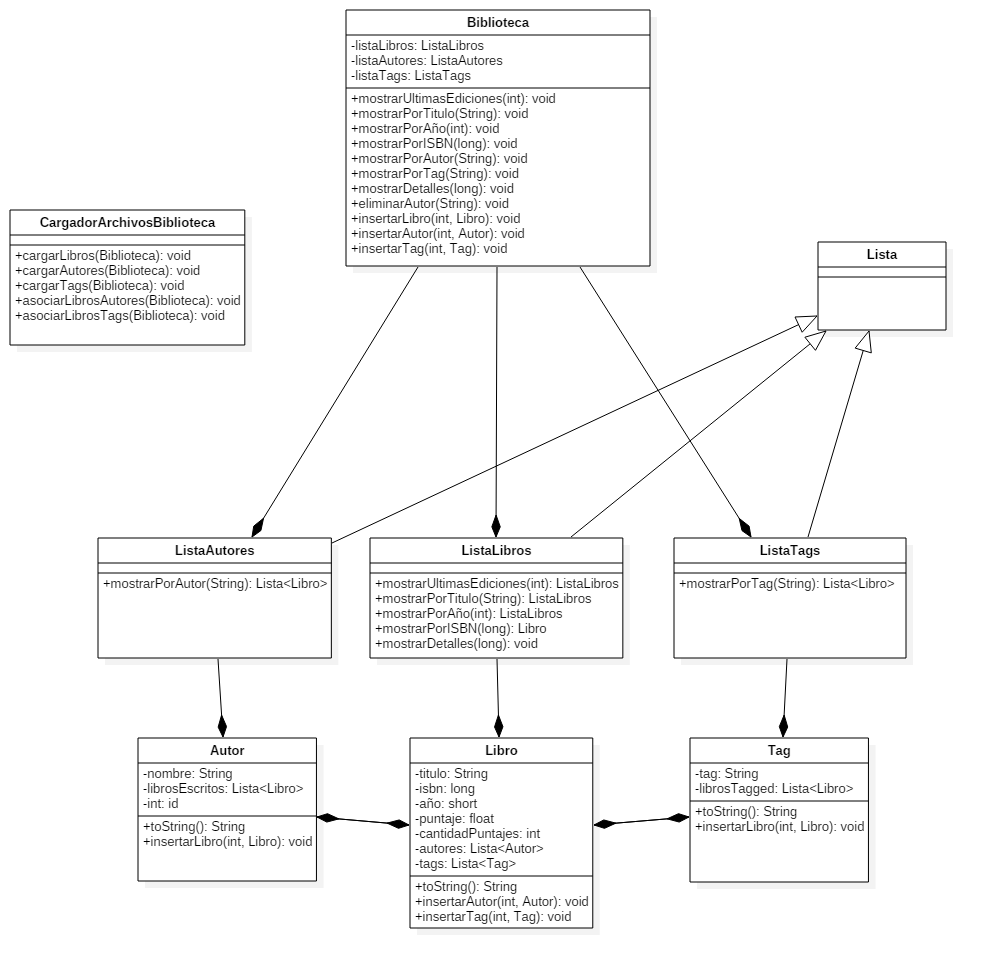
Teniendo en cuenta que **t >> n > m** es evidente que la mejor alternativa es la segunda (la que implementa árboles). Aunque el costo de memoria es un poco más alto, los órdenes de tiempo de ejecución decrecen drásticamente en comparación a la alternativa 1.

Los siguientes valores aproximados fueron obtenidos utilizando como referencia los archivos provistos, y suponiendo que las proporciones de las entradas se mantienen iguales a medida que crecen:

|  |  |
| --- | --- |
| **Funcionalidades requeridas** | **Alternativa 2 en comparación a alternativa 1** |
| Buscar por título | 460 veces mas rápido |
| Buscar por año o ISBN | Igual |
| Buscar por tag | 2200 veces más rápido |
| Buscar por autor | 400 veces más rápido |
| Buscar últimas ediciones | Igual |
| Mostrar detalles de un libro | Igual |
| Eliminar autor y sus libros | 460 veces más rápido |
| Eliminar un tag | 2200 veces más rápido |

# Selección y justificación de alternativa a implementar

La alternativa seleccionada para este problema fue una ligera modificación de la segunda. A continuación se puede ver el diagrama de clases completo:



El cambio fue que se agregaron las clases ListaLibros, ListaAutores y ListaTags. Estas tres clases heredan del TDALista utilizado (que aparece en blanco en el diagrama dado que ya todos conocemos sus operaciones). La clase biblioteca ya no realiza las operaciones requeridas por su cuenta, sino que delega la mayoría de ellas a una de esas tres clases. Como la lógica de todas estas operaciones no se altera con respecto a las presentadas en seudocódigo y su orden de tiempo de ejecución es igual, no es necesario desarrollarlas.

Lo que más se tomó en cuenta para decidir esta alternativa fue los órdenes de tiempo de ejecución de las operaciones de búsqueda, dado que esto es lo que más se realiza en una biblioteca real. El hecho de que la primera alternativa consuma un poco menos de memoria no es tan importante en comparación a todo el tiempo que se ahorra con cada búsqueda que se realiza.

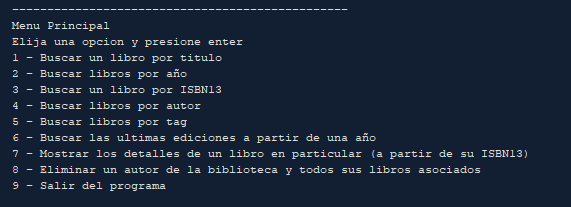
También se tuvo en cuenta que, en cuanto a la escalabilidad del sistema, se va a notar mucho más la diferencia entre los tiempos de ejecución de ambos algoritmos a medida que crece la base de datos (por ejemplo, si para un cierto tamaño de base de datos, buscar un libro por su tag demora 10 segundos en la segunda alternativa, en la primera alternativa demoraría varios días).

# Conclusiones

En conclusión, se escogió la segunda alternativa como la mejor solución. Ésta tiene además la ventaja de que es muy fácil de implementar y mantener dada la simplicidad de sus algoritmos.

# Guía de usuario

1. Abrir el entorno de desarrollo NetBeans.
2. Abrir el proyecto “proyecto\_AED1”.
3. Correr la clase “Proyecto\_AED1.java”. En este momento se comienzan a cargar los archivos que la biblioteca utiliza como base de datos. Este proceso demora aproximadamente 3 minutos.
4. Cuando se terminan de cargar los archivos, se podrá ver el siguiente menú:



1. Ingrese el numero correspondiente a la operación que desee realizar y presione enter.
2. Ingrese el criterio para la búsqueda/eliminación y nuevamente presione enter.
3. Si la búsqueda fue exitosa, podrá ver en pantalla los libros que coinciden con el criterio dado anteriormente. Si realizó una eliminación, simplemente se le informará si fue exitosa o no. Luego de esto se le mostrará de nuevo el menú principal para que pueda escoger otra opción.
4. Cuando desee salir del programa, simplemente elija la opción “9” en el menú principal.

1. Se toma como máximo 5 autores por libro (pocos libros sobrepasan este máximo, pero se complementa con la gran mayoría que no lo alcanza) [↑](#footnote-ref-1)
2. Tomando como referencia los archivos de la base de datos, se toma como máximo 150 tags por libro. [↑](#footnote-ref-2)
3. Se toma como máximo 10 libros por autor, dados los archivos [↑](#footnote-ref-3)
4. Se toma como máximo 10 libros por tag, dados los archivos [↑](#footnote-ref-4)
5. Se toma como máximo 5 autores por libro (pocos libros sobrepasan este máximo, pero se complementa con la gran mayoría que no lo alcanza) [↑](#footnote-ref-5)
6. Tomando como referencia los archivos de la base de datos, se toma como máximo 150 tags por libro. [↑](#footnote-ref-6)
7. Se toma como máximo 10 libros por autor, dados los archivos [↑](#footnote-ref-7)
8. Se toma como máximo 10 libros por tag, dados los archivos [↑](#footnote-ref-8)