

Algoritmos y Estructuras de Datos I

**PROYECTO INDIVIDUAL – PARTE I**

Francisco Piria

# Contenido

[Contenido 2](#_Toc512786711)

[Introducción 3](#_Toc512786712)

[Problema planteado 3](#_Toc512786713)

[Análisis de alternativas 3](#_Toc512786714)

[Primera alternativa 3](#_Toc512786715)

[Segunda alternativa 6](#_Toc512786716)

[Algoritmos 8](#_Toc512786717)

[Primera alternativa 8](#_Toc512786718)

[Segunda alternativa 15](#_Toc512786719)

[Comparación de alternativas 20](#_Toc512786720)

[Selección y justificación de alternativa a implementar 21](#_Toc512786721)

[Conclusiones 21](#_Toc512786722)

# 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:

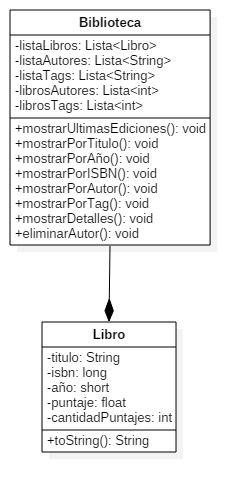
* Permitir hacer búsquedas de libros por nombre, autor, tag, año o ISBN y mostrar en pantalla los resultados
* Mostrar en pantalla los detalles de un libro en particular
* Mostrar las últimas ediciones a partir de un cierto año
* Eliminar un autor y todos sus libros

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.

### Primera alternativa

La primera alternativa tiene la siguiente estructura:

La clase Biblioteca, que será la encargada de realizar las tareas de UcuBooks, contiene 5 listas. Cada una de esas listas se corresponde con uno de los archivos de la base de datos de UcuBooks. Una de las listas contiene objetos Libro y las etiquetas de sus nodos son los ID de los libros. Otra lista contiene strings que representan a los autores (con etiqueta ID para cada autor). Otra hace lo mismo para los tags que puede tener un libro, y las últimas dos establecen una conexione entre IDs de libros y ID de autores y tags respectivamente.

En cuanto a los costos de memoria de esta alternativa, a continuación se ve el costo de un objeto Libro:

|  |  |
| --- | --- |
| **Lista de libros** | **Espacio** |
| Titulo - String[[1]](#footnote-1) | 100 B |
| ISBN - long | 8 B |
| Año - short | 2 B |
| Puntaje - float | 4 B |
| Cantidad de puntajes - int | 4 B |
| ID - int | 4 B |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **126 B** |

Este sería el espacio en memoria ocupado por cada libro que se agrega a la biblioteca. Luego también hay que tener en cuenta la memoria ocupada por los contenidos de las otras listas. Estos son los costos por autor y por tag:

|  |  |
| --- | --- |
| **Lista de autores** | **Espacio** |
| Nombre - String[[2]](#footnote-2) | 60 B |
| ID - int | 4 B |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **68 B** |

|  |  |
| --- | --- |
| **Lista de tags** | **Espacio** |
| tag - String[[3]](#footnote-3) | 60 B |
| ID - int | 4 B |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **68 B** |

Ahora solo falta analizar el costo de las dos listas de asociación:

|  |  |
| --- | --- |
| **Lista libros-autores** | **Espacio** |
| ID libro - int | 4 B |
| ID autor - int | 4 B |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **12 B** |

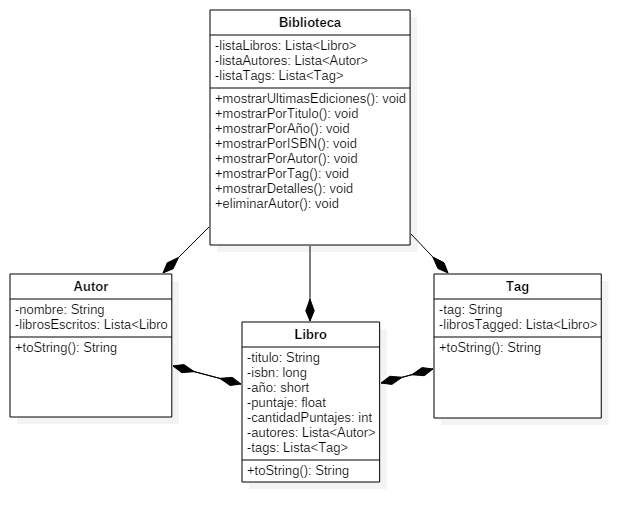
|  |  |
| --- | --- |
| **Lista libros-tags** | **Espacio** |
| ID libro - int | 4 B |
| ID tag - int | 4 B |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **12 B** |

Para conocer un estimado del espacio total requerido por el programa para esta alternativa, utilizamos los tamaños de los archivos que forman la base de datos de UcuBooks.

|  |  |
| --- | --- |
|  | **Espacio** |
| Lista de libros – aprox. 5800 elementos | 730800 B |
| Lista de autores – aprox. 4800 elementos | 326400 B |
| Lista de tags – aprox. 34000 elementos | 2312000 B |
| Lista libros-autores – aprox. 8400 elementos | 100800 B |
| Lista libros-tags – aprox. 586000 elementos | 7032000 B |
| **Total** | **10502000 B = 10,5 MB** |

Como podemos ver, la escalabilidad de esta alternativa (en cuanto a la memoria) no es muy buena dado que a medida que crece la cantidad de libros y tags, la cantidad de asociaciones entre ellos crece mucho más rápido. Esta gran cantidad de asociaciones es lo que más ocupa lugar en el programa.

### Segunda alternativa

La segunda alternativa al problema tiene esta estructura:

Para esta alternativa, se agregaron dos clases que representan las entidades 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, solo 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 | 100 B |
| ISBN - long | 8 B |
| Año - short | 2 B |
| Puntaje - float | 4 B |
| Cantidad de puntajes - int | 4 B |
| Lista de referencias a autores[[4]](#footnote-4) | 20B nodos + 20B referencias a siguientes |
| Lista de referencias a tags[[5]](#footnote-5) | 600B nodos + 600B referencias a siguientes |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **1362 B** |

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

|  |  |
| --- | --- |
| **Lista de autores** | **Espacio** |
| Nombre - String | 60 B |
| Lista de referencias a libros[[6]](#footnote-6) | 40B nodos + 40B referencias a siguientes |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **144 B** |

|  |  |
| --- | --- |
| **Lista de tags** | **Espacio** |
| Tag - String | 60 B |
| Lista de referencias a libros[[7]](#footnote-7) | 40B nodos + 40B referencias a siguientes |
| Referencia en el nodo al siguiente en la lista | 4 B |
| **Total** | **144 B** |

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

|  |  |
| --- | --- |
|  | **Espacio** |
| Lista de libros – aprox. 5800 elementos | 7899600 B |
| Lista de autores – aprox. 4800 elementos | 691200 B |
| Lista de tags – aprox. 34000 elementos | 4896000 B |
| **Total** | **13486800 B = 13,5 MB** |

En este caso la memoria requerida también crece rápidamente, debido a la gran cantidad de referencias entre todos los objetos que maneja el programa.

## Algoritmos

Antes de comenzar las descripciones en seudocódigo y análisis de ordenes de tiempo de ejecución para las funcionalidades requeridas, seria adecuado explicar brevemente las operaciones del TDA Lista, dado que ambas alternativas implementan este tipo de datos.

Las operaciones mas 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)

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

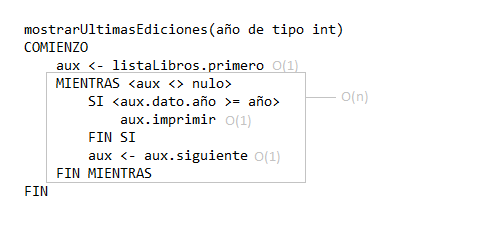
##### Precondiciones

* 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 de tiempo de ejecución para 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

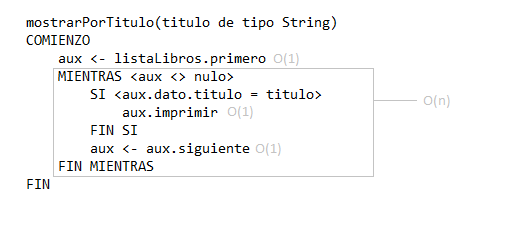
##### Precondiciones

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

##### 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 tiempo de ejecución de este algoritmo es de O(n)

#### mostrarPorAño

##### Lenguaje Natural

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

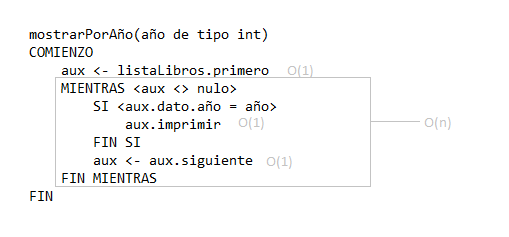
##### Precondiciones

* 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 tiempo de ejecución de este algoritmo es O(n).

#### mostrarPorISBN

##### Lenguaje Natural

Recorre la lista de libros e imprime el libro cuyo ISBN es igual al ISBN dado

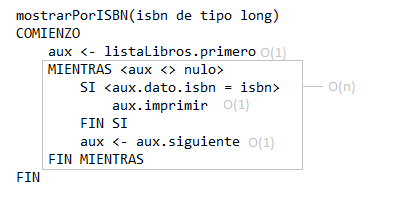
##### Precondiciones

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

##### 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 tiempo de ejecución de este algoritmo es O(n).

#### mostrarPorAutor

##### Lenguaje Natural

Recorremos primero la lista de autores y nos quedamos con el ID del autor cuyo nombre coincide con el nombre dado. Luego recorremos la lista que asocia libros a autores y nos quedamos con los IDs de libros que están asociados al ID de autor que teníamos. Finalmente, buscamos en la lista de libros esos IDs e imprimimos los libros que coinciden.

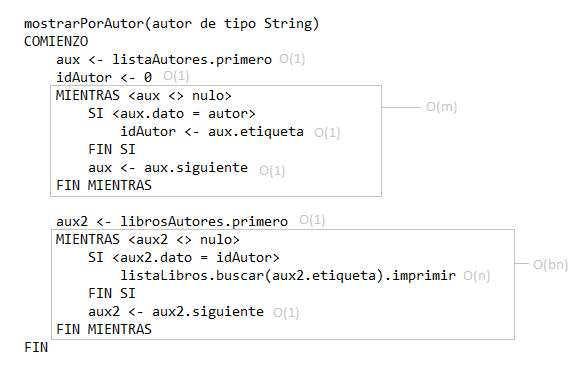
##### Precondiciones

* La lista de autores contiene una **m** cantidad de nodos, cada nodo contiene un String que representa el nombre del autor, y las etiquetas de los nodos son el ID de cada autor.
* La lista de asociaciones libro-autor contiene una **b** cantidad de nodos, cada nodo contiene un int que representa un ID de autor, y su etiqueta es el ID del libro asociado.
* La lista de libros contiene una **n** cantidad de nodos, cada nodo contiene un libro y su etiqueta es su ID.

##### Postcondiciones

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

##### Seudocódigo



Siendo “m” el tamaño de la lista de autores y “b” el tamaño de la lista libros-autor, este algoritmo resulta ser de orden O(bn), dado que tanto b y n son mayores que m.

#### mostrarPorTag

##### Lenguaje Natural

Recorremos primero la lista de tags y nos quedamos con el ID del tag cuyo nombre coincide con el nombre dado. Luego recorremos la lista que asocia libros a tags y nos quedamos con los IDs de libros que están asociados al ID de tag que teníamos. Finalmente, buscamos en la lista de libros esos IDs e imprimimos los libros que coinciden.

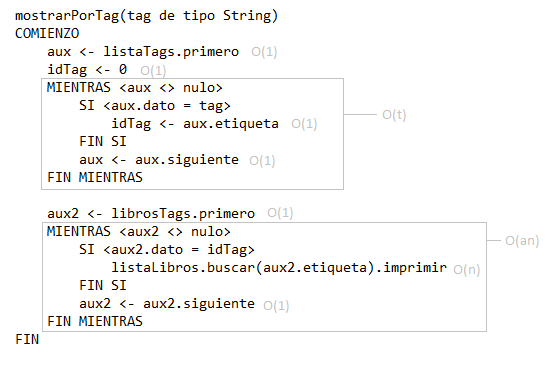
##### Precondiciones

* La lista de tags contiene una **t** cantidad de nodos, cada nodo contiene un String que representa el nombre del tag, y las etiquetas de los nodos son el ID de cada tag.
* La lista de asociaciones libro-tag contiene una **a** cantidad de nodos, cada nodo contiene un int que representa un ID de tag, y su etiqueta es el ID del libro asociado.
* La lista de libros contiene una **n** cantidad de nodos, cada nodo contiene un libro y su etiqueta es su ID.

##### Postcondiciones

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

##### Seudocódigo



Siendo “t” el tamaño de la lista de tags y “a” el tamaño de la lista libros-tag, este algoritmo resulta ser de orden O(an), dado que a es mucho mayor que t.

#### mostrarDetalles

##### Lenguaje Natural

Primero buscamos el ID dado en la lista de libros e imprimimos el libro. Luego buscamos en la lista libro-autor todos los IDs de autor que están asociados al ID de libro y utilizamos esos IDs de autor para buscar en la lista de autores todos los autores del libro y los imprimimos. Finalmente realizamos lo mismo para los tags del libro.

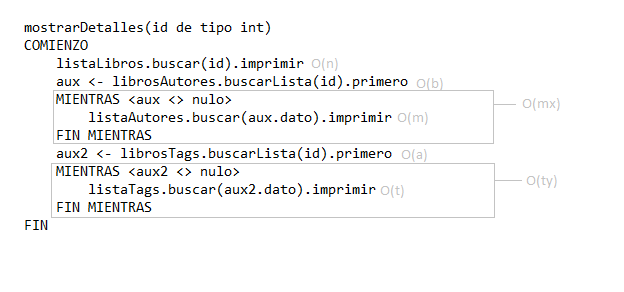
##### Precondiciones

* La lista de tags contiene una **t** cantidad de nodos, cada nodo contiene un String que representa el nombre del tag, y las etiquetas de los nodos son el ID de cada tag.
* La lista de asociaciones libro-tag contiene una **a** cantidad de nodos, cada nodo contiene un int que representa un ID de tag, y su etiqueta es el ID del libro asociado.
* La lista de autores contiene una **m** cantidad de nodos, cada nodo contiene un String que representa el nombre del autor, y las etiquetas de los nodos son el ID de cada autor.
* La lista de asociaciones libro-autor contiene una **b** cantidad de nodos, cada nodo contiene un int que representa un ID de autor, y su etiqueta es el ID del libro asociado.
* La lista de libros contiene una **n** cantidad de nodos, cada nodo contiene un libro y su etiqueta es su ID.
* Existe un libro en la biblioteca que tiene el ID dado.

##### Postcondiciones

* Ninguna de las listas se modificó
* Se imprimió en pantalla toda la información del libro que coincide con el ID dado, en caso de haberlo

##### Seudocódigo



Se le llamo “x” a la cantidad de autores que tiene el libro buscado e “y” a la cantidad de tags que tiene. Dado que estas son cantidades pequeñas en comparación a los tamaños de las listas que se manejan, se puede tomar al primer “mientras” como de orden O(m) y al segundo como de orden O(t). Entonces por la regla de la suma el orden de todo el algoritmo es el mayor de los siguientes ordenes: O(n), O(b), O(m), O(a) y O(t). Tomando como referencia los archivos provistos, el orden del algoritmo es entonces O(a).

#### eliminarAutor

##### Lenguaje Natural

Primero buscamos el autor dado en la lista de autores, lo eliminamos y nos quedamos con su ID. Luego buscamos en la lista libro-autor todos los IDs de libros que están asociados al ID de autor, los quitamos de la lista y luego utilizamos esos IDs para eliminar en la lista de libros todos los libros de ese autor. Finalmente eliminamos las asociaciones del autor eliminado a sus libros utilizando su ID.

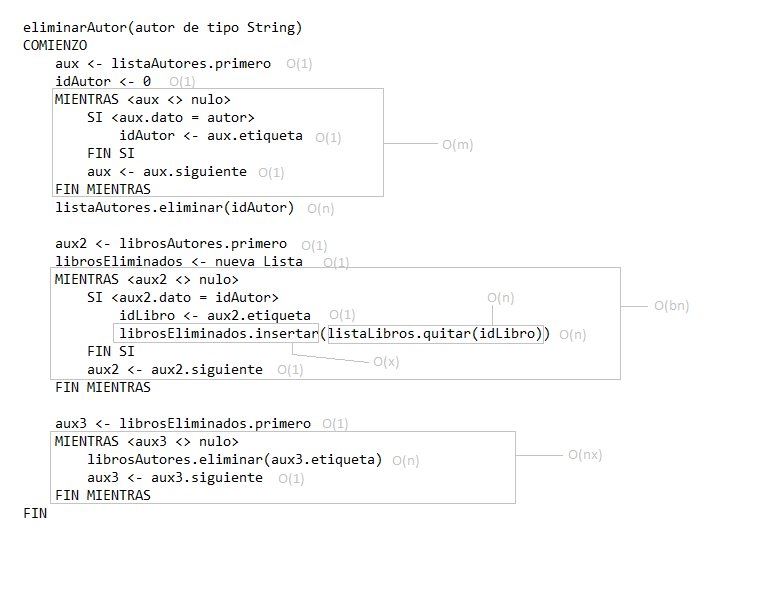
##### Precondiciones

* La lista de autores contiene una **m** cantidad de nodos, cada nodo contiene un String que representa el nombre del autor, y las etiquetas de los nodos son el ID de cada autor.
* La lista de asociaciones libro-autor contiene una **b** cantidad de nodos, cada nodo contiene un int que representa un ID de autor, y su etiqueta es el ID del libro asociado.
* La lista de libros contiene una **n** cantidad de nodos, cada nodo contiene un libro y su etiqueta es su ID.

##### Postcondiciones

* Se eliminó al autor dado por parámetro (si existía en la biblioteca) y la lista de autores ahora contiene un elemento menos.
* Se eliminaron todos los libros del autor en la lista de libros.
* Se eliminaron todas las asociaciones entre el autor y sus libros en la lista libros-autores.

##### Seudocódigo



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

### Segunda alternativa

#### mostrarUltimasEdiciones

##### Lenguaje Natural

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

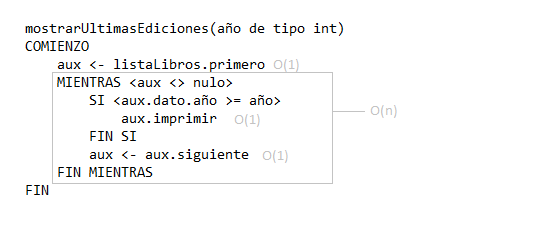
##### Precondiciones

* 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

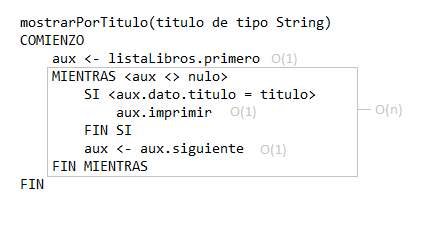
##### Precondiciones

* 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

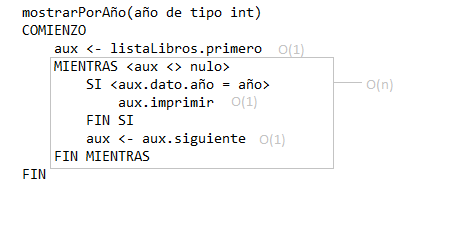
##### Precondiciones

* 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.

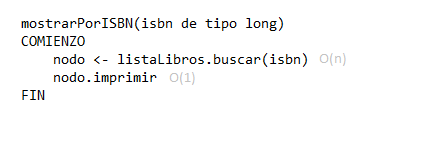
##### Precondiciones

* 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.

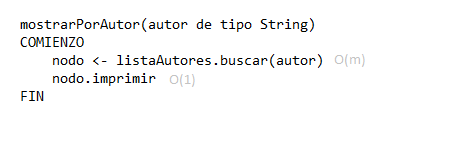
##### Precondiciones

* 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.

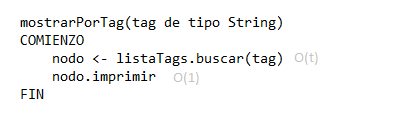
##### Precondiciones

* 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 algorimo 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).

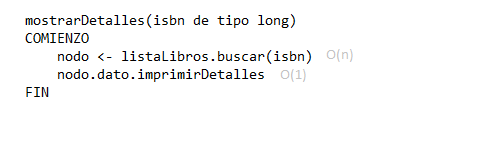
##### Precondiciones

* 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.

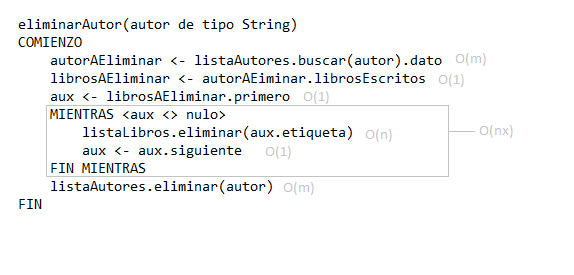
##### Precondiciones

* La lista de autores 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).

### 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, año o ISBN | O(n) | O(n) |
| Buscar por tag | O(an) | O(t) |
| Buscar por autor | O(bn) | O(m) |
| Buscar últimas ediciones | O(n) | O(n) |
| Mostrar detalles de un libro | O(a) | O(n) |
| Eliminar autor y sus libros | O(bn) | O(n) |
|  |  |  |
| Memoria requerida | 10,5 MB | 13,5 MB |

Teniendo en cuenta que **a >> t > b > n > m** es evidente que la mejor alternativa es la segunda (la que define clases Autor y Tag). Aunque el costo de memoria es un poco mas alto, los ordenes de tiempo de ejecución decrecen drásticamente en comparación a la alternativa 1.

Los siguientes valores fueron obtenidos utilizando como referencia los archivos provistos:

|  |  |
| --- | --- |
| **Funcionalidades requeridas** | **Alternativa 2 en comparación a alternativa 1** |
| Buscar por título, año o ISBN | Igual |
| Buscar por tag | 100000 veces más rápido |
| Buscar por autor | 10000 veces más rápido |
| Buscar últimas ediciones | Igual |
| Mostrar detalles de un libro | 100 veces más rápido |
| Eliminar autor y sus libros | 8400 veces más rápido |

# Selección y justificación de alternativa a implementar

La alternativa seleccionada fue la segunda. Lo que mas se tomó en cuenta para esta decisión fue los ordenes de tiempo de ejecución de las operaciones de búsqueda, dado que esto es lo que mas 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

Resumen de las características más relevantes del producto. ¿Por qué debería el cliente elegir mi solución?

1. Para el titulo se tomó como tamaño arbitrario un máximo de 50 caracteres. Dado que muy pocos libros sobrepasan este máximo, se compensa con la gran mayoría de los títulos que no lo alcanzan, y aun así, 50 sigue siendo un máximo bastante generoso [↑](#footnote-ref-1)
2. Para los autores se tomó un tamaño máximo de 30 caracteres, por las mismas razones que los títulos [↑](#footnote-ref-2)
3. Para los tags se tomó un tamaño máximo de 30 caracteres, por las mismas razones [↑](#footnote-ref-3)
4. 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-4)
5. Tomando como referencia los archivos de la base de datos, se toma como máximo 150 tags por libro. [↑](#footnote-ref-5)
6. Se toma como máximo 10 libros por autor, dados los archivos [↑](#footnote-ref-6)
7. Se toma como máximo 10 libros por tag, dados los archivos [↑](#footnote-ref-7)