

Algoritmos y Estructuras de Datos I

**PROYECTO INDIVIDUAL – PARTE I**

Francisco Piria

Contenido

[Introducción 3](#_Toc417411916)

[Problema planteado 3](#_Toc417411917)

[Análisis de alternativas 3](#_Toc417411918)

[Algoritmos 3](#_Toc417411919)

[Selección y justificación de alternativa a implementar 3](#_Toc417411920)

[Conclusiones 3](#_Toc417411921)

[Guía del usuario 3](#_Toc417411922)

# 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:

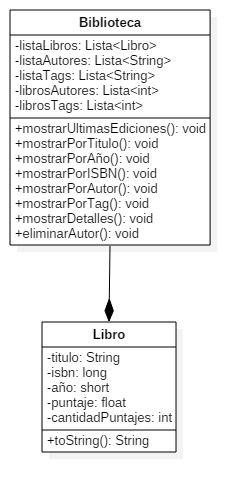
* Permitir hacer búsquedas de libros por nombre, autor, genero, año o ISBN y mostrar en pantalla los resultados
* Mostrar en pantalla los detalles de un libro en particular
* Mostrar las últimas ediciones a partir de un cierto año
* Eliminar un autor y todos sus libros

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 sera 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 ultimas 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 |

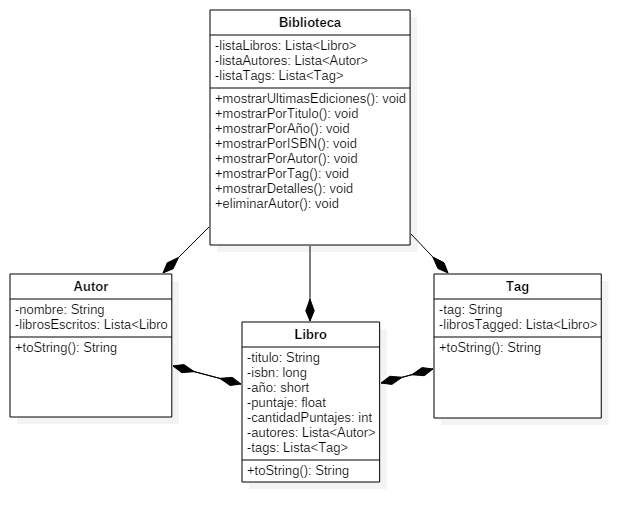
|  |  |
| --- | --- |
| Lista libros-tags | Espacio |
| ID libro - int | 4 B |
| ID tag - int | 4 B |
| Referencia en el nodo al siguiente en la lista | 4 B |
| Total | 12 B |

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 mas 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

* Descripción en seudocódigo (de acuerdo a los estándares de la asignatura) de las diferentes funcionalidades que el programa debe cumplir, para cada alternativa de representación indicada arriba.
* Análisis del orden del tiempo de ejecución de cada funcionalidad, para cada alternativa de representación indicada arriba.

# Selección y justificación de alternativa a implementar

Breve fundamentación de la elección.

Descripción detallada de la alternativa implementada (dibujos, diagramas, aclaraciones sobre clases e interfaces).

# Conclusiones

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

# Guía del usuario

* Instrucciones (breves) para el uso de la aplicación.
* Consideraciones de instalación (si corresponde) y requerimientos no funcionales para la operación.

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 tomo 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)