Facultad de Ciencias Exactas

Tecnicatura Universitaria en Desarrollo de Aplicaciones Informáticas

Trabajo Práctico Especial

Primera Etapa

"Colección de libros – Implementación con Java"

Meliendrez Agustín
Santos, Luciano

Tandil, Mayo 2018

1. Introducción

A partir de la cátedra “Programación 3”, perteneciente la carrera “Tecnicatura Universitaria en Desarrollo de Aplicaciones Informáticas” (TUDAI), de la Facultad de Ciencias Exactas, perteneciente a la Universidad Nacional del Centro de la Provincia de Buenos Aires, se planteó la problemática de implementar, en Java, una herramienta la cual permite simplificar la búsqueda de libros según un género (arte, terror, comedia, etc.).

Es por ello, que, a partir del siguiente informe, se presentarán las discusiones realizadas para la búsqueda de las mejores implementaciones posibles, en base a las estructuras conocidas y enseñadas por la cátedra.

Una vez planteada la discusión de las diferentes opciones disponibles, se presentará cuál fue la opción seleccionada para llevar a cabo el desarrollo del programa, como así también el por qué de su selección.

En tercer lugar, se presentarán los ensayos de pruebas realizadas, a través de los resultados obtenidos, para demostrar si la implementación seleccionada era la correcta, o si se encontraron errores en su desarrollo.

Por último, se presentarán las conclusiones alcanzadas a través del desarrollo del Trabajo Práctico Especial, como así también dificultades o limitaciones encontradas en el desarrollo del mismo.

2. Desarrollo

Antes de comenzar con la implementación del código, para resolver el problema planteado, se esquematizó cómo debía funcionar el programa. A partir de ello, se determinaron, en principio, dos clases que debían existir,

En primer lugar se encuentra la clase “Libro”, que será la que contiene los datos particulares de cada libro (como por ejemplo título, autor, etc.).

En segundo lugar está la clase “Índice”, que es la que posee los “géneros” de los libros (arte, terror, etc.), de manera ordenada, y referencias a los libros pertenecientes a cada género.

Una vez planteados que clases principales debían existir, y antes de comenzar a determinar cómo estaban conformadas dichas clases, se plantearon que estructuras se vieron en la cátedra, y así ver las ventajas y desventajas de cada una, para luego realizar las elecciones que se creían más óptimas en cada caso.

Las estructuras que se tuvieron en cuenta para el desarrollo del trabajo fueron:

- ***Array***: Esta estructura tiene como ventaja su rápido acceso si se conoce en qué posición esta un elemento determinado, o si se quiere insertar un nuevo elemento, y se posee la posición del último elemento. Sin embargo, su tamaño es estático, por lo que, si se llega al límite, se debe crear un nuevo "array" con más capacidad y realizar el traspaso de la información. Por otro lado, si se lo quiere mantener ordenado, la complejidad de ordenamiento y espacio en memoria varía según lo que se haya seleccionado. Por ejemplo, si se utiliza el algoritmo "Merge sort" la complejidad será de Log_2 n, por lo que su ordenamiento será más rápido, pero utilización de memoria será O(n). Por otro lado, si se utiliza un algoritmo de "Burbujeo" (o Bubblesort), su complejidad será $O(n^2)$, pero su utilización de memora será $O(n^2)$.

- ***ArrayList***: Es una lista que se implementa a partir de arrays. Su tamaño se va modificando dinámicamente, pero tiene un costo adicional (ya que debe resolver el problema de límite de un array). La complejidad de las funcionalidades varía. Por ejemplo, para insertar un elemento al final es O(1), pero si se quiere insertar en una posición determinada o se quiere eliminar su complejidad es O(n).

- ***LinkedList***: Es una lista implementada a partir de lo que se denominan nodos y punteros o referencias a otros nodos. Su tamaño, al igual que el arrayList, se modifica dinámicamente. Insertar un elemento al principio o final es O(1), pero si se quiere obtener un elemento específico su complejidad es O(n), ya que siempre se deben recorrer todos los elementos hasta llegar a uno específico. Se consideró desde un principio descartar la desarrollada por la cátedra y utilizar a la provista por java, por el hecho de que ésta posee los métodos optimizados, para reducir los tiempos de procesamiento.

- ***Árbol Binario de Búsqueda:*** Su tamaño varía dinámicamente, según se vayan agregando nuevos "nodos". Todas sus operaciones tienen un costo algorítmico de O(log n), siempre y cuando se considere a un árbol que este balanceado.

Una vez planteadas dichas estructuras se pasó a la selección de cada una en base los requerimientos solicitados.

2.1 Clase Libro:

La clase libro contiene la información individual de cada libro. Entre ellas se encuentran:

- Título,
- Autor,
- Cantidad de Páginas,
- Géneros.

Una vez determinada las variables necesarias, se planteó de que “tipo” debían ser cada una.

Con respecto a las primeras tres, se determinó que debían ser del tipo “String”, ya que no constituía el núcleo del problema a resolver, presentando una información única por variable, y solo iba a ser solicitado el título del autor.

Por su parte, la primera dicotomía surgió en la selección de la variable a utilizar para los “géneros”, producto de que un libro podía contener múltiples géneros diferentes, y donde, a su vez, no todos los libros poseen la misma cantidad de géneros.

De esta manera se seleccionó entre las estructuras de array, arraylist y linkedlist, ya que en todos los libros era necesario recorrer todos los elementos. En esta selección el árbol se descartó inmediatamente, ya que siempre iba a ser necesario recorrer todos los géneros de un libro.

Luego, se descartó usar un arreglo porque la cantidad de géneros de un libro puede variar entre ellos, y se estaría reservando espacio en memoria que nunca sería utilizado.

Finalmente, entre “ArrayList” y “LinkedList” se optó por la segunda, debido a que ésta posee la ventaja en insertar elementos. A su vez, si bien la primera tiene la ventaja de acceder a un elemento particular, esto no iba a ser necesario ya que siempre se debía recorrer todos los elementos.

2.2 Clase Índice

Para esta clase, se partió de la premisa de que se debía seleccionar la mejor implementación que permitiera la optimización de la búsqueda de libros según géneros.

Por otro lado, debía presentar una segunda estructura que retornara todos los libros del género solicitado, utilizando referencias a los libros que estarían almacenados en memoria.

Para la selección de la estructura general de la clase índice se seleccionó desde un comienzo un "Árbol Binario de Búsqueda". Si bien el árbol no tiene mecanismos de balanceo implementados, se consideró que, si estuvieran implementados, se mejoraría la eficiencia de las búsquedas e inserciones frente a otras estructuras, ya que, según el problema planteado, se debía permitir las búsquedas constantes de un género determinado y, en caso de no existir, agregarlo de forma que se mantenga el índice ordenado.

En este sentido, se descartó la LinkedList, ya que para la búsqueda de un género, en el peor de los casos, era O(n), a diferencia del árbol que posee una complejidad de O(Log n). Por su parte, se descartaron el array y el arraylist, ya que el tamaño del índice puede variar según la cantidad de géneros que se presenten y, en este aspecto, se consideró que el árbol otorgaba mayor flexibilidad en la inserción y mayor eficiencia de búsqueda, ya que para lograr la misma complejidad en el array o arrayList, primero hay que mantenerlo ordenado, hecho que se vuelve "costoso" en grandes cantidades de información.

Por otro lado, una vez planteada la estructura general del índice, se consideró que cada "nodo" del árbol debía poseer la información de:

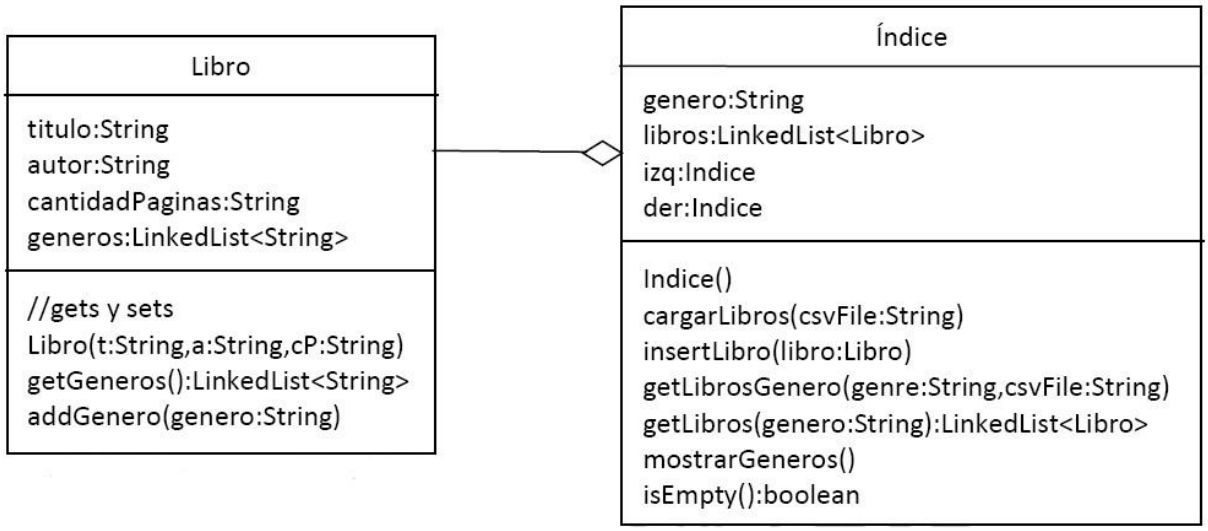
- Nombre del género.
- Lista de libros del género.

Para el nombre se utilizó el tipo "String", ya que los géneros son únicos.

Por su parte, para la lista de libros del género, se utilizó una "LinkedList" que posee referencias a los libros pertenecientes a dicho género.

La selección de una "LinkedList" fue debido a que, en primer lugar no era necesario que este ordenada, y por lo tanto su inserción poseía una complejidad de O(1). En segundo lugar, se debían devolver todos los libros del género, por lo que en cualquier estructura seleccionada se tendría una complejidad obligatoria de O(n) para recorrer la estructura y devolver la información de todos los libros existentes. Por último se seleccionó frente a otras estructuras (como el array), por la flexibilidad que otorga con respecto su tamaño, ya que no todos los géneros tendrán la misma cantidad de libros.

Una vez planteada las dos clases, se puede decir que el diagrama de clases planteado seria de la siguiente manera:



3. Resultados

Para evaluar la eficacia de la implementación, se realizaron se realizaron una serie pruebas para observar el comportamiento en la carga de datos en memoria, como a su vez la creación de un archivo de salida utilizando distintas métricas. Para ello, se utilizaron los archivos provistos por la catedra “dataset1.csv”, “dataset2.csv”, “dataset3.csv” y “dataset4.csv”, los cuales contenían colecciones de libros de diferentes tamaños.

Para la primera prueba se utilizó la clase Timer provista por la catedra para calcular el tiempo (en segundos) que tardaba el programa en resolver los dos casos. Para la carga de datos en memoria se realizaron 5 pruebas distintas y se promediaron los resultados. Para la creación, y descarga del archivo, se utilizaron los géneros drama, terror, humor, moda y novela, y se sacó un promedio con los resultados obtenidos. En este caso los datos obtenidos fueron los siguientes:

Imagen 1: elaboración propia

Imagen 2: elaboración propia

Se puede observar que tanto para la carga como para la creación del archivo de salida el tiempo aumenta dependiendo el tamaño del archivo de entrada (archivo “csv” seleccionado) y el archivo de salida que se quiere generar.

En la segunda prueba realizada, se calculó la cantidad de iteraciones (dentro de un bucle “for”) que se realizan para cargar los libros en memoria y cuantas se realizan para crear el archivo de salida utilizando siempre para el mismo el género de terror.

Imagen 3: elaboración propia

Imagen 4: elaboración propia

Como se observa en los gráficos, tanto para la carga como para la descarga del archivo de salida, las iteraciones aumentan dependiendo la cantidad de libros de la colección de libros provistas.

Finalizando, en la última prueba se calculó la cantidad de nodos que se recorren en el árbol utilizado como índice para agregar los géneros en la carga y para encontrar los géneros cuando se quiere generar el archivo de salida.

Imagen 5: elaboración propia

Imagen 6: elaboración propia

Como se ve en los gráficos, dependiendo la cantidad de libros va a aumentar o disminuir la cantidad de nodos visitados. En cambio, cuando se busca el género para generar el archivo de salida, la cantidad de nodos visitados es similar para todos los casos.

Sin embargo, al no estar balanceado el árbol va a depender de en qué momento se cargó el género terror para el resultado, como así también de la cantidad de géneros existentes. Si el árbol estuviera balanceado el resultado para todos los casos seria casi idéntico. Es por ello que en los gráficos

presentados, se puede observar que en el “Dataset 1” hay más cantidad de nodos visitados que en el “Dataset 4”.

4. Conclusión

Para concluir el trabajo, se puede destacar algunas limitaciones encontradas, como así también algunas modificaciones realizadas a lo largo de la elaboración del programa.

En primer lugar, cuando se planteó el problema, y se comenzó a codificar el programa, éste poseía una “LinkedList” en la cual se guardaba la colección entera de libros. Sin embargo, se observó que solo se lograba ocupar más espacio en memoria, el cual era desperdiciado, y el cual no aportaba nada a la solución. A su vez, en el planteo del problema no se pedía en ningún momento listar todos los libros de la colección, por lo que se eliminó dicha lista, manteniendo las referencias en el índice.

En segundo lugar, y desprendiéndose del anterior párrafo, al eliminarse dicha lista, la solución presenta la limitación de que, si posteriormente se quieren agregar funcionalidades al mismo, como buscar un libro en particular, al solo tener las referencias en el índice, la eficiencia del mismo se vería afectada.

Por último, al observar todos los resultados obtenidos podemos ver que los mismos dependen en casi todos los casos de la cantidad de libros que se cargan, y que si el árbol estaría balanceado la búsqueda del genero para generar el archivo de salida sería muy eficiente, ya que no dependería de la cantidad de libros. Por lo tanto, la principal limitación del programa es que no se implementó un método para mantener balanceado el árbol.