

**TPE Colección de libros**

primera etapa

TUDAI | Programación 3 | 04-05-2018

Autor

Magalí Molina (maguimarchena@gmail.com)

Institución



# **Introducción**

El objetivo es realizar las clases y métodos necesarios para construir una colección de libros partiendo de un archivo csv. La colección debe tener un índice organizado por géneros para facilitar la búsqueda de libros por los distintos géneros a los que están asociados. El resultado de estas búsquedas debe ser volcado en un nuevo archivo csv.

Luego de la implementación se debe analizar distintas métricas de la performance al trabajar con archivos de 20, 1.000, 100.000 y 1.000.000 de libros respectivamente.

**Desarrollo**

Elección de estructuras

Las 3 estructuras posibles para la implementación tienen un costo de operaciones distinto, específicamente en sus métodos para insertar elementos, ordenarlos y acceder al elemento contenido en una posición (índice) ó nodo.

LinkedList y ArrayList de Java tienen una complejidad O(1) para agregar un elemento al final, mientras que la implementación de un Árbol Binario de Búsqueda cuenta con una complejidad O(log n) para la inserción. La ventaja del ABB, además de una complejidad menor es que la inserción se realiza de forma ordenada por lo que no requiere un ordenamiento posterior.

Pasando a la complejidad de ordenamiento y búsqueda, ArrayList tiene O(n log n) para ordenamiento (sort()) y O(1) para búsquedas (get()); LinkedList con O(n) para ordenar y O(n/2) para buscar, mientras que ABB tiene O(n log n) en búsqueda y no requiere ordenamiento.

ArrayList tiene ventaja sobre ABB en inserción y búsqueda, e inversamente, ABB tiene una clara ventaja en ordenamiento ya que se realiza conjuntamente con la inserción. Si bien el ordenamiento consume una porción importante de tiempo, para este caso sólo se realiza una vez en el índice al finalizar el agregado de libros a la colección, mientras que la búsqueda se realiza en múltiples oportunidades.

Tanto para la colección como el índice, elegí usar ArrayList principalmente por la facilidad de implementación que brinda al ser una clase de Java, probada y actualizada de forma regular, con una amplia variedad de métodos ya establecidos.

Pruebas de ejecución

Se realizaron 2 tipos de pruebas, de tiempos y de cantidad de iteraciones.

En las pruebas de tiempo, implementadas con la clase Timer provista por la cátedra, se ve claramente la influencia del tamaño del archivo en el tiempo total de ejecución.

Se realizaron dos sets de pruebas, uno para la carga de la estructura y otro para la búsqueda de libros por género. En ambos sets se llevan a cabo cuatro pruebas, una por cada tamaño de archivo, con 5 repeticiones. [[1]](#footnote-1)

Para un tamaño de archivo de 20, los tiempos son de 0.008 segundos para la carga y de 0.001 segundos para la búsqueda, en promedio, con máximos de 0.018 s. carga y 0.0016 s. búsqueda y mínimos de 0.0014 s. carga y 0.00013 s. búsqueda.

Para tamaño 1.000 los valores no aumentan en gran medida, mientras que para tamaño 100.000 se observa un leve incremento. El aumento más importante se ve, como se esperaba, en el tamaño 1.000.000.

**Pruebas de carga:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Carga | | | |
|  | 20 | 1.000 | 100.000 | 1.000.000 |
| valor 1 | 0,017559 | 0,052151 | 1,023500 | 45,082704 |
| valor 2 | 0,001771 | 0,059814 | 1,048894 | 0,506631 |
| valor 3 | 0,016936 | 0,048873 | 1,038633 | 2,600351 |
| valor 4 | 0,001908 | 0,050718 | 1,029887 | 15,426910 |
| valor 5 | 0,001504 | 0,049638 | 0,102081 | 15,734213 |
| **Max** | 0,017559 | 0,059814 | 1,048894 | 45,082704 |
| **Min** | 0,001504 | 0,048873 | 0,102081 | 0,506631 |
| **Promedio** | 0,007936 | 0,052239 | 0,848599 | 15,870162 |

**Pruebas de búsquedas de libros:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Búsqueda | | | |
|  | 20 | 1.000 | 100.000 | 1.000.000 |
| valor 1 | 0,001326 | 0,002254 | 0,021949 | 0,815325 |
| valor 2 | 0,001594 | 0,002526 | 0,022869 | 0,531739 |
| valor 3 | 0,001348 | 0,000241 | 0,021300 | 0,139799 |
| valor 4 | 0,000126 | 0,001971 | 0,020351 | 1,125213 |
| valor 5 | 0,000136 | 0,002173 | 0,021607 | 0,625386 |
| **Max** | 0,001594 | 0,002526 | 0,022869 | 1,125213 |
| **Min** | 0,000126 | 0,000241 | 0,020351 | 0,139799 |
| **Promedio** | 0,000906 | 0,001833 | 0,021615 | 0,647492 |

Para las pruebas de cantidad de iteraciones, se creó la clase IterationCounter y se realizaron nuevamente dos sets de pruebas, igual a las anteriores, para analizar la carga de la estructura y la búsqueda en los cuatro tamaños de archivo

**Conclusión**

Como puede verse en las tablas y gráficos, ambos procesos son afectados por el tamaño del archivo. La carga de la estructura representa una tarea compleja que requiere más tiempo, sin embargo, este proceso se realizará sólo en escasas ocasiones para aumentar la colección mientras que las búsquedas serán repetidas múltiples veces y no requieren más que un par de segundos para generar el archivo de salida en colecciones muy grandes y apenas unas milésimas para colecciones más acotadas.

1. Aclaración: el archivo de tamaño 1.000.000 no se encuentra en el repositorio de github ya que el mismo no permitió la subida debido al peso del archivo. [↑](#footnote-ref-1)