Facultad de Ciencias Exactas

Tecnicatura Universitaria en Desarrollo de Aplicaciones Informáticas

Programación 3

Trabajo Práctico Especial – 1er entrega

“Buscador de Libros por género”

Alumnos:

Falcón Matias - mfalcon@alumnos.exa.unicen.edu.ar

Notti Agustina - anotti@alumnos.exa.unicen.edu.ar

Tandil, 3 de junio del 2022

ÍNDICE

INTRODUCCIÓN

El siguiente informe se presenta en el marco de la cátedra *Programación 3*, perteneciente al segundo año de la carrera Tecnicatura Universitaria en Desarrollo de Aplicaciones Informáticas (TUDAI), de la Facultad de Ciencias Exactas (UNICEN).

El mismo, tiene como fin explicar el proceso de análisis e implementación del Trabajo Práctico Especial (TPE) planteado por la cátedra y las conclusiones a las que se arribó.

La problemática planteada para el desarrollo del mismo consiste en la implementación de una herramienta, en lenguaje Java, que permita simplificar la búsqueda de libros por géneros, partiendo de una colección de libros, donde cada uno de ellos está compuesto por un titulo, autor, una cantidad de paginas y un conjunto de géneros que describen el contenido, como pueden ser *arte*, *ciencia*, *policial*, entre otros.

El desarrollo del Trabajo Práctico Especial se llevará a cabo en dos etapas:

En esta primera etapa, se llevará a cabo la implementación de la lógica para obtener una colección de libros que contenga un género en particular, ingresado por el usuario.

La herramienta comenzará llevando a memoria la colección completa de libros para luego realizar un filtrado por un género dado, presentando al usuario la colección de libros resultante.

Para optimizar el proceso de búsqueda, se requiere implementar un índice por género, el cual simplificará el acceso a solo un subconjunto de todos los libros existentes.

Consignas

- Discutir y analizar el costo de las operaciones sobre las distintas estructuras que se podrían utilizar para almacenar los libros en memoria. Justificar la elección de la estructura considerando la utilización de la misma dentro de la herramienta.
- Considerando las siguientes estructuras como posibles índices de acceso por género:
    - Una lista simplemente vinculada (implementada en el práctico 1).
    - Alguna de las implementaciones conocidas de la interfaz List de Java.
    - Un árbol binario de búsqueda.

  Discutir y analizar el costo para realizar un filtrado utilizando cada una de las estructuras propuestas como índice de acceso.

- Implementar el índice de acceso por géneros eligiendo una de las estructuras. Justificar su elección.
- Realizar pruebas de ejecución utilizando los distintos .cvs provistos y analizar los resultados, considerando distintas métricas (tiempo de ejecución, cantidad de iteraciones, cantidad de nodos visitados, etc).

Teniendo en cuenta las consignas mencionadas, el informe busca reflejar el análisis y discusiones que se llevaron a cabo para arribar a una solución de la problemática planteada, como así también las pruebas y resultados obtenidos.

DESARROLLO

1. Estructuras de almacenamiento de datos

1.1 Tipos de Estructuras de Datos

Existen diferentes tipos de estructuras que son utilizadas para almacenar y organizar datos, las cuales resultan más eficientes o no, dependiendo de la problemática que se quiera resolver.

Para el desarrollo de este trabajo, se analizaron las siguientes estructuras:

- **Array:** Los arreglos son estructuras de datos estáticas, es decir, son bloques contiguos en memoria que una vez creados, su tamaño no puede modificarse. Por lo tanto, de requerir más espacio es necesario generar un nuevo arreglo y hacer el traspaso de todos los elementos hacia el arreglo de mayor tamaño.

  Si bien es un tipo de estructura que tiene como ventaja el rápido acceso a memoria tanto para obtener un elemento del cual conocemos su posición, como para insertar un nuevo elemento al final(O(1)), la operación de búsqueda de un elemento, de inserción en una posición específica o de eliminación de un elemento va a depender de la cantidad de elementos que contenga el array, ya que en el peor de los casos va a tener que recorrer n elementos para realizar tales operaciones (O(n)).

- **ArrayList:** los Arraylist son una estructura que permite almacenar datos de manera similar a los arrays, con la diferencia de que para estas no es necesario definir su tamaño inicialmente, es decir, es un tipo de estructura dinámica. Sin embargo, los arraylist tienen un costo adicional ya que internamente estos deben resolver la limitación de espacio de los arrays.

  Al igual que los arrays, las diferentes operaciones que se pueden realizar sobre un ArrayList varían en complejidad. Mientras que para obtener un elemento de una posición específica o insertar un elemento al final la complejidad es constante (O(1)), para buscar o eliminar un elemento es necesario recorrer todos sus elementos (en en el peor de los casos), por lo tanto su complejidad es O(n).

- **LinkedList:** Una lista vinculada es una estructura de datos compuesta por nodos, donde cada uno de ellos, además del valor o información específica, está compuesto por un puntero que hace referencia a su nodo siguiente.

A diferencia de los arreglos, una lista vinculada no tiene un tamaño fijo ni se almacena en un bloque continuo en memoria, sino que su tamaño es variable y la asignación de memoria para sus elementos se genera en tiempo de ejecución. Esto hace que para acceder a cualquier elemento de la lista haya que conocer la ubicación del nodo principal y luego, a partir de él, recorrer todos los elementos hasta llegar al elemento buscado, lo que hace que para la operación de búsqueda la complejidad computacional sea O(n). Sin embargo, para insertar un elemento al inicio de la lista la complejidad es O(1).

- **Árbol Binario de Búsqueda (ABB):** Un árbol es un tipo de estructura de datos que tiene un carácter recursivo, ya que está compuesto por nodos, donde existe un nodo raíz y los restantes son considerados subárboles. Para cada uno de estos subárboles, el nodo raíz está conectado por medio de un arco.

  Un ABB es un tipo particular de árbol binario (cuyos nodos no van a tener más de dos hijos) que cumple con la siguiente propiedad: *Para los nodos del ABB, todas las claves del subárbol izquierdo son menores que la de la raíz, y todas las claves del subárbol derecho son mayores a la de la raíz.*

  Para las operaciones de inserción y borrado la complejidad es O(h) donde h es la altura del árbol (la longitud de la rama más larga), ya que en el peor de los casos tiene que ir hasta el último nodo de la rama más larga. Sin embargo, para la búsqueda de un elemento, la complejidad va a ser O(log n), ya que la propiedad de los ABB mencionada permite descartar ciertos elementos a la hora de comenzar la búsqueda, siempre y cuando sea un árbol balanceado.

1.2 Almacenamiento de Libros en Memoria

Para el almacenamiento de los libros en memoria, decidimos utilizar una Lista Vinculada, ya que, en principio, no sabiendo la totalidad de elementos que vamos a almacenar, es más eficiente gracias a que no tiene un tamaño fijo. De esa forma, salvamos la limitación de espacio en el caso de que usaramos un array o arraylist.

Por otro lado, la Lista Vinculada posee ventajas en cuanto a la inserción de elementos al inicio, por sobre los Array o ArrayList.

A su vez, como no necesariamente se necesitaba que los elementos estén ordenados, optamos también por descartar el Árbol Binario de Búsqueda.

1.3 Índice de Acceso por Género

Para el Índice de Acceso por Género, decidimos usar un Árbol Binario de Búsqueda ya que éste permite realizar una búsqueda eficiente, descartando ciertos elementos al inicio de la operación (siempre y cuando el árbol esté balanceado), sin necesidad de recorrer todos sus nodos para hacer un filtrado. Se descartaron el ArrayList y la Lista Vinculada ya que, para realizar la operación de filtrado por género, sería necesario recorrer todos los elementos, aumentando la complejidad y el costo computacional.

A su vez, en caso de no encontrar el género buscado, la estructura de ABB permite insertar los elementos de manera ordenada según la clave del nodo.

Por último, el ABB también posee ventajas en cuanto a su dinamismo de tamaño. Ya que no se sabe de antemano la cantidad de elementos a insertar, el ABB permite ir generando el espacio en memoria necesario para cada nuevo elemento.

2. Diseño e implementación

Para la implementación de la herramienta se definieron 3 clases principales: Indice, Género y Libro.

2.1 Clase Libro

La clase *Libro* contiene los datos principales de cada libro, como *título*, *autor* y *cantidad de páginas*, atributos que se definieron del tipo String.

También, cada libro posee una lista de géneros, la cual se definió del tipo LinkedList, por las ventajas mencionadas en apartados anteriores.

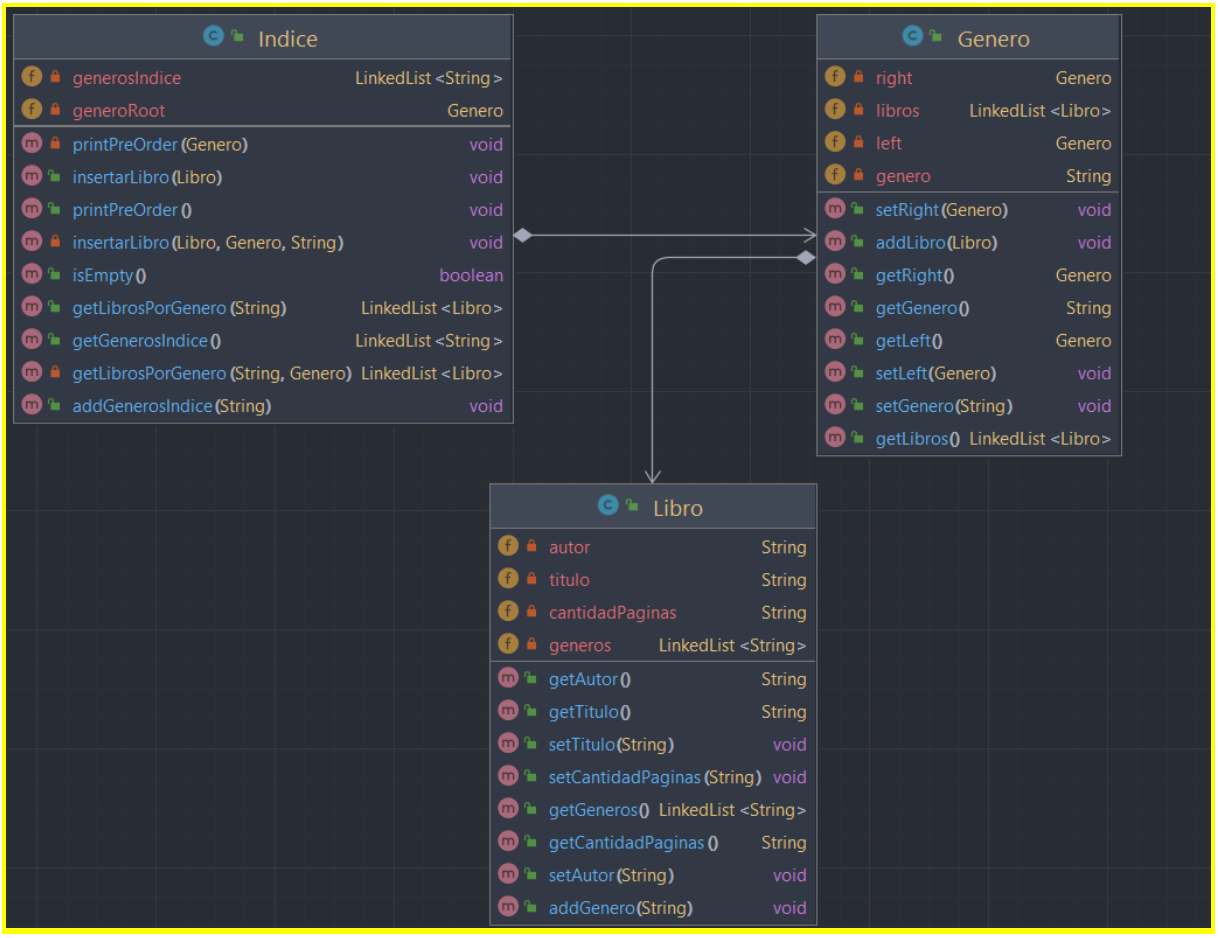
2.2 Clase Índice

La clase *Índice* es la que está implementada en base a la estructura de Árbol Binario de Búsqueda, permitiendo mayor eficiencia tanto en el filtrado de los libros como en la inserción de los nuevos nodos.

Cada nodo del Índice está compuesto por un *Género* y una lista de géneros que conforman el Índice, que se va actualizando con cada nueva inserción. Esta lista de géneros es utilizada para mostrar al usuario las posibilidades de géneros que puede ingresar para la búsqueda.

2.3 Clase Género

La clase *Género* compone el nodo del Índice. Cada *Género* está compuesto por el nombre del Género (del tipo String), una lista de libros (del tipo LinkedList), con referencias a los libros que pertenecen a cada *Género*, y referencias a sus nodos izquierdo y derecho (del tipo *Género*).



Resultados obtenidos y Conclusiones

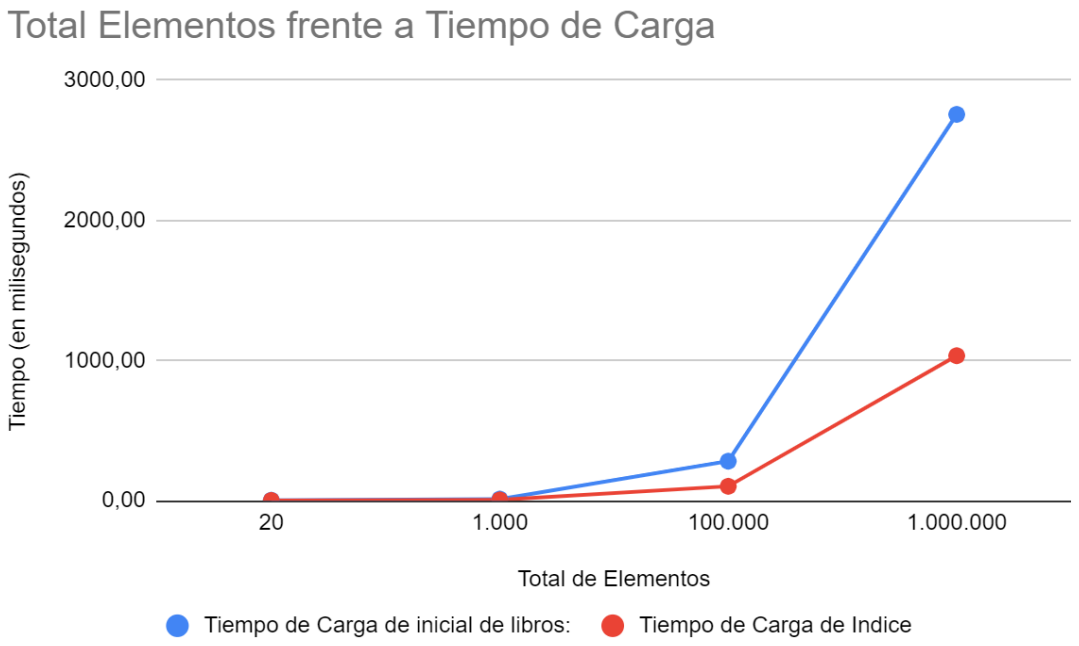
Luego de la implementación de la herramiento realizamos diferentes pruebas para evaluar el comportamiento de la misma, teniendo en cuenta diferentes volúmenes de datos, a partir de archivos provistos por la cátedra para tal fin.

Para evaluar el comportamiento de la herramienta se consideraron 3 momentos del proceso: la carga de libros inicial, la carga del índice y la búsqueda de libros por género. A su vez, se tuvo en cuenta el *tiempo* y la *cantidad de elementos* como métricas de comparación.

A partir de las pruebas realizadas, se llegó a las siguientes conclusiones:

- El *Tiempo de Carga Inicial* y *Tiempo de Carga de Índice* es directamente proporcional a la cantidad de elementos.

- El *Tiempo de Carga Inicial* es mayor que el *Tiempo de Carga de Índice* ya que en el primero está contemplado el tiempo que se consume en generar un nuevo libro con todos los datos que se observan en el archivo y luego agregarlo a la lista.

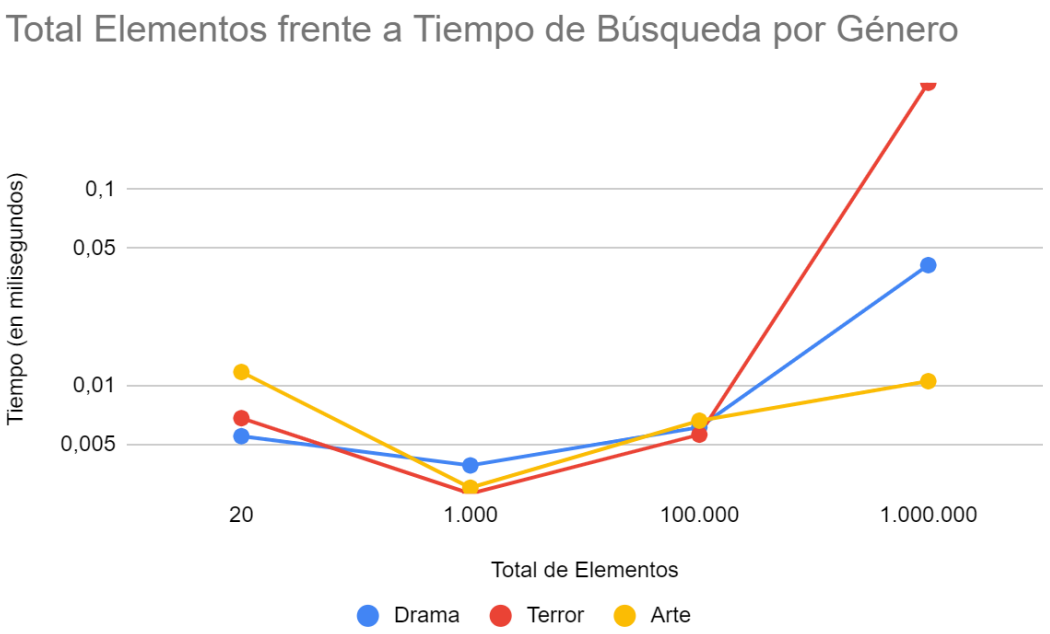


- El *Tiempo de Búsqueda por Género* es variable ya que no depende de la cantidad de elementos que haya sino de la cantidad de géneros que tenga agregados el índice, como así también del orden en que son cargados los elementos, por las características propias de la estructura del Árbol Binario de Búsqueda. Situación similar sucede con la cantidad de nodos visitados.

  Recordemos que en la implementación realizada, el árbol no tiene aplicados mecanismos de balanceo. Entendemos que si estos mecanismos fueran implementados, los tiempos serían similares para cada búsqueda.

  Para graficar este comportamiento se utilizó como ejemplo la búsqueda de libros para los Géneros *Drama*, *Terror* y *Arte*.

Total Elementos	**Tiempo de Búsqueda por Género**		
	Drama	**Terror**	**Arte**
20	0,0055	0,0068	0,0117
1.000	0,0039	0,0028	0,003
100.000	0,0061	0,0056	0,0066
1.000.000	0,0411	0,3508	0,0105


Total Elementos frente a Tiempo de Búsqueda por Género
Tiempo (en milisegundos)
0,1
0,05
0,01
0,005
20
1.000
100.000
1.000.000
Total de Elementos
Drama
Terror
Arte