Informe

Trabajo Práctico Especial

Entrega Nro 1

Programación III

TUDAI

Universidad Nacional Del Centro de la Provincia de Buenos Aires

Integrantes:

- Mazza, Eloy
  - eloy.mazza@gmail.com
- Muñoz, Silvina
  - silmunoz22@gmail.com

Fecha de Entrega: 04/05/2018

Introducción

La propuesta brindada por la cátedra para realizar este trabajo práctico especial es lograr el desarrollo de una herramienta que permita realizar la búsqueda de libros por un género dado por el usuario.
Para ello se solicitó la implementación de la herramienta en código Java y un informe sobre la funcionalidad y estructura elegida, donde también se presentarán los resultados obtenidos en cada prueba con los distintos volúmenes de datos. La cátedra nos brinda archivos .CSV con los datos necesarios para poder realizar dichas pruebas. Es por esto que se analizará el tiempo de ejecución de cada prueba y a su vez se realizarán gráficos donde se refleje el comportamiento de la estructura elegida en cada una de las pruebas.

Desarrollo del trabajo:

- ## Discusiones y análisis:

  En primer lugar, se analizó de qué manera se almacenarán los libros en memoria. Dado que contábamos con la clase ListaArreglo ya implementada, decidimos utilizar dicha estructura para el almacenamiento de los libros. Esta estructura consta de varios métodos, pero el más utilizado en nuestro caso es el método *insert(Object o)*, en donde se chequea que al momento de insertar un nuevo objeto, en este caso un nuevo Libro, el arreglo no esté completo, dado que si esta completo, duplica su tamaño. A su vez, en esta clase tenemos implementado el método *binarySearch()* que nos sera muy útil al momento de realizar una búsqueda debido a su complejidad temporal O(n).

  Al momento de decidir de qué manera almacenar los libros según el género que le corresponda, se discutió sobre las siguientes estructuras:

  - Una lista simplemente vinculada (implementada en el práctico 1).
  - Alguna de las implementaciones conocidas de la interface List de Java.
  - Un árbol binario de búsqueda.

  La primer y segunda estructura al momento de insertar como de buscar un género, tienen una complejidad O(n), es decir, va a recorrer tantos nodos como los que tenga para llegar al último e insertar uno nuevo, y lo mismo sucede con la búsqueda. En este caso no nos sirve ya que tendría un costo muy alto al momento de leer los archivos de entradas que contengan una gran cantidad de libros.
  La última estructura, el árbol binario fue la que utilizamos. Dentro de esta insertamos géneros, ordenados alfabéticamente, lo cual nos facilitara al momento de insertar y/o buscar. La complejidad temporal utilizando esta estructura es de Log2(n), y en comparación con la estructuras anteriores podemos ver que disminuirá nuestro costo al momento de insertar o buscar dentro de nuestro árbol.

- **Decisiones de diseño:**

  Se optó por implementar tres clases principales: “LibrarySystem”, que representa la clase que posee la estructura de almacenamiento (“Library”), y que también posee la forma de cargar los libros en memoria a partir de los archivos .csv provistos, y escribir los resultados de las búsquedas de libros por géneros, ambas funcionalidades encapsuladas en la clase “Scanner".

  Se decidió tener separado en clases distintas la estructura de la forma de obtener los datos para aumentar el encapsulamiento y permitir cambiar las formas de almacenar e importar/exportar con mayor facilidad. Así, la clase “LibrarySystem” se comunica con estas dos clases mediante sus respectivas interfaces, y nunca ingresa a su estructura interna.

  A su vez, la clase Library está compuesta por una instancia de la clase “ListaArreglo” (Implementación propia de la clase ArrayList) de “Books” (Clase que representa a cada libro) y una instancia de la clase “GenderTree”, (Clase propia que hereda de “BinaryTree”, implementación propia de árbol binario de busqueda). Se decidió utilizar la clase ListaArreglo para almacenar los libros ya que como no hay inconvenientes en insertarlos al final una vez creados, el coste de tiempo es o(1) (Se modificó el método insertar para lograr esta complejidad). Por otra parte, se genero la clase “GenderTree”, que es un árbol binario de búsqueda cuyos nodos son instancias de la clase Género. Así, el tiempo de acceso a los géneros es O(log2n). A su vez cada género posee una lista de punteros hacia los libros que les pertenecen. Esta estructura también permite realizar una búsqueda binaria de libros por género, mediante el método searchBooks() que recibe el nombre del género que se va a usar para retornar los libros que pertenezcan al mismo, con una complejidad temporal O(log2n).

  La clase Scanner posee un método llamado “importBooks”, que recibe un “path” (que se puede pasar en el constructor o setear mediante su respectivo setter) y una instancia de “Library”, que es en donde se van a guardar los libros que se van obteniendo de los archivos “database$.csv”.

Luego, para exportar, se programó el método printBooksByGeneder, que recibe una lista de libros filtrada por géneros obtenido del “GenderTree”, y un “path” que es la ubicación del archivo en el cual se van a grabar estos libros.

- **Resultados obtenidos:**

Tiempo:

- **Cargar libros en memoria:**
  - **Dataset1:** 0.0054 segundos
  - **Dataset2:** 0.0241 segundos
  - **Dataset3:** 1.2767 segundos
  - **Dataset4:** 12.8647 segundos

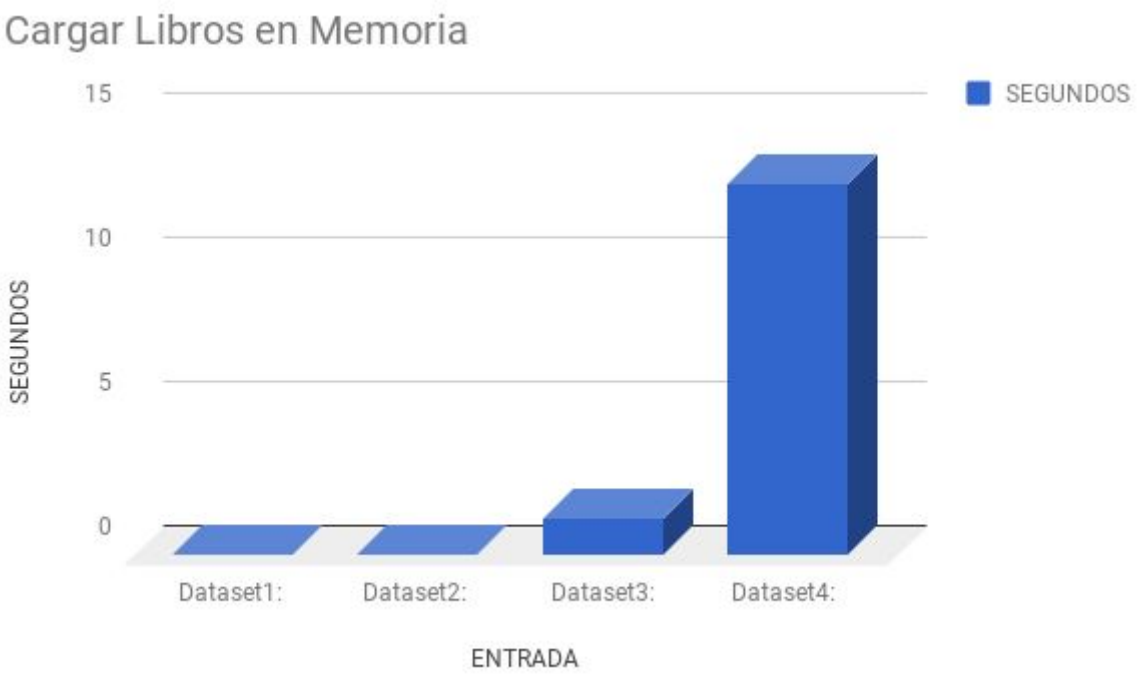
- **Buscar y Escribir datos en archivo (Promedio):**
  - **Dataset1**: 0.00001 segundos
  - **Dataset2**: 0.0007 segundos
  - **Dataset3**: 0.0213 segundos
  - **Dataset4**: 0.4094 segundos

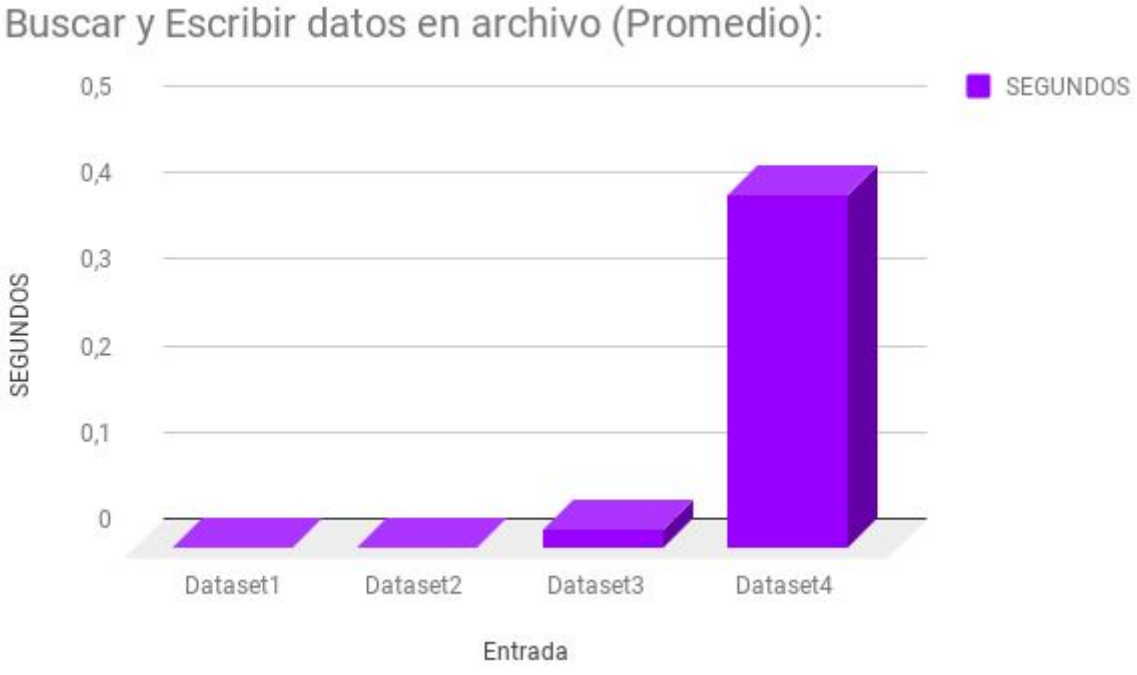
Cantidad de nodos visitados:

- **Generar Indice libros:**
  - **Dataset1:** 468
  - **Dataset2:** 32,409
  - **Dataset3:** 2,841,717
  - **Dataset4:** 27,656,790

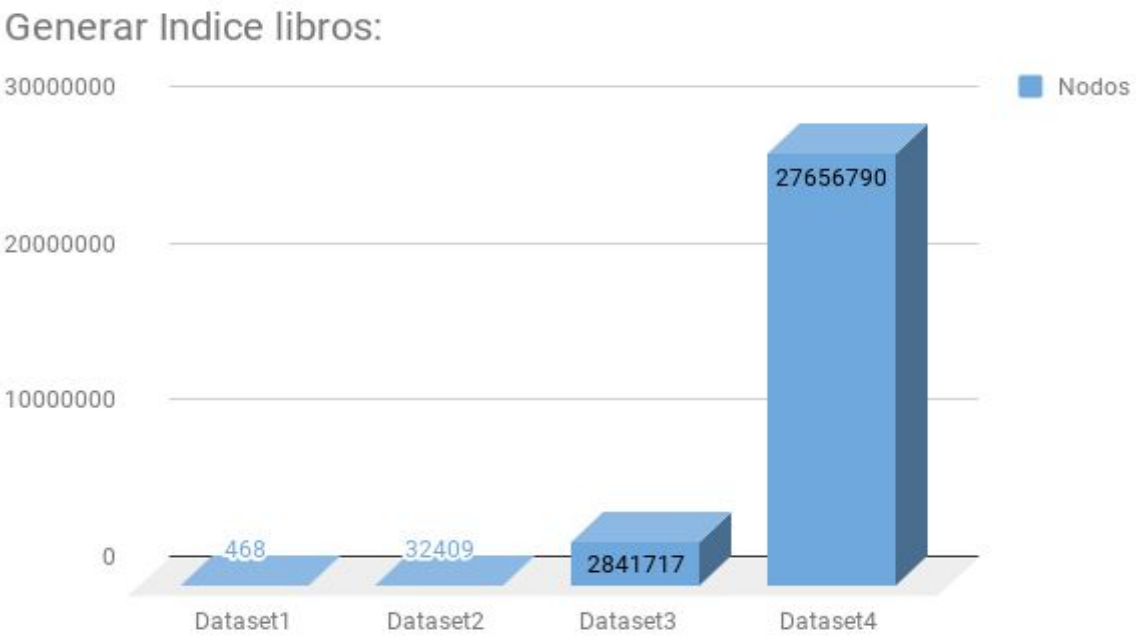
- **Encontrar lista libro dado un genero (Promedio)**:
  - **Dataset1:** 21
  - **Dataset2:** 20
  - **Dataset3:** 20
  - **Dataset4:** 24

- **Gráficos:**
  - **Gráficos de Tiempo:**

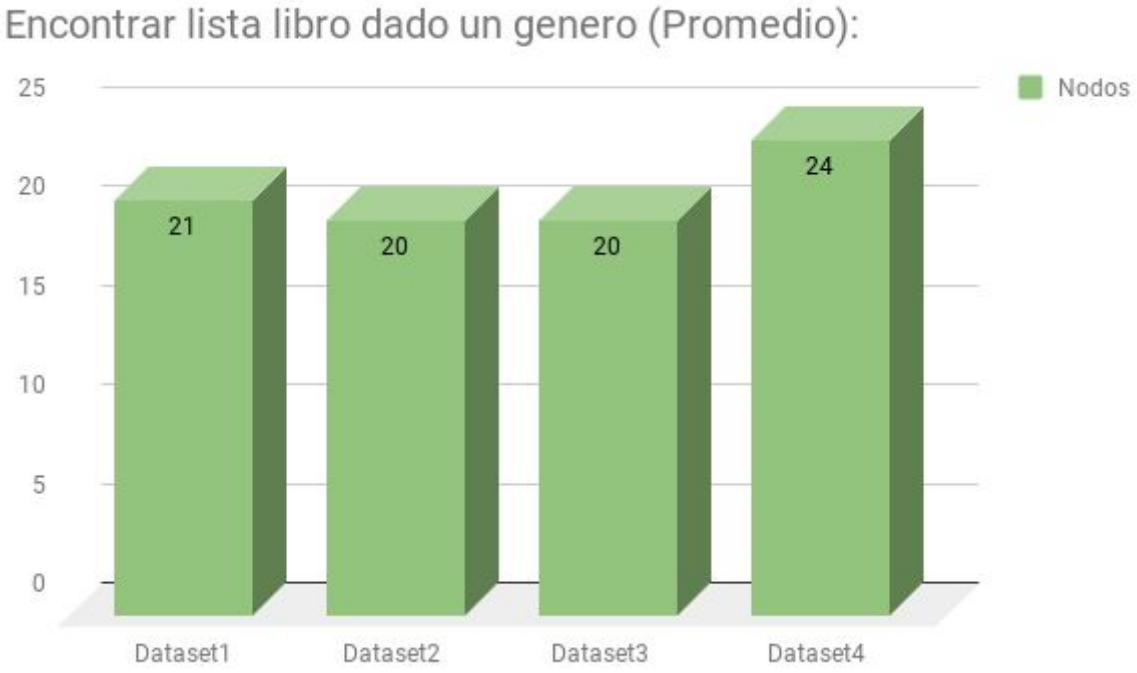




- **<u>Gráficos de paso por nodos:</u>**



- **Encontrar lista libro dado un genero (Promedio)**:



Conclusiones:

Según los datos obtenidos al analizar tiempo de ejecución y paso por nodos llegamos a las siguientes conclusiones:

- El tiempo de importar desde el archivo, crear los libros, almacenarlos y generar el indice por géneros (12 segundos en el caso del dataset mayor) es mucho mayor que el de buscar la lista de libros filtrando por algún género en particular (en promedio medio segundo para buscar y escribir el archivo). Cabe aclarar que el aumento del tiempo en el apartado de "**Buscar y Escribir datos"** se debe a que cada vez tarda más en escribir una mayor cantidad de datos, ya que buscar lo hace casi instantáneamente.
- El tiempo en almacenar los libros es O(1) (Constante), ya que los almacenamos en un arreglo uno después del otro, utilizando el ID del libro para acceder con velocidad constante al índice correcto donde se debe guardar el libro.
- Las veces que se tiene que iterar para generar el índice de libros por género es: L (cantidad de libros) * N(g/l) (Cantidad de géneros que contenga ese libro) * Log2N (iteraciones para encontrar el Género en el árbol)
- Buscar la lista de libros dado un género es muy veloz, requiere visitar pocos nodos, ya que la búsqueda en un árbol binario es (log2N).
- Insertar nuevos géneros en el árbol es veloz, (log2N).
- Agregar un libro a la LinkedList de libros de cada Género es constante si se inserta al comienzo, O(N) si se hace al final.