Trabajo Práctico especial Programación 3

“Índice de búsqueda por géneros”

*Primera entrega*

Materia: Programación 3

Carrera: T.U.D.A.I. Correspondiente a la facultad de Ciencias Exactas (UNICEN)

*Autores.*

Belen Sofia Enemark

Mail:belenenemark@gmail.com

Matias Ezequiel Gonzalez

Mail: monatios@gmail.com

**Introducción**

Partiendo de una colección de libros, se desea implementar una herramienta que permita simplificar la búsqueda de libros por géneros, para lo cual se deberán implementar de forma lógica y justificada las estructuras. Las mismas permiten darle eficiencia a la creación del “*índice de búsqueda*” y a la estructura utilizada para almacenar los libros. Las mismas serán testeadas para demostrar sus ventajas/desventajas frente a otras con la finalidad de optimizar de la mejor forma posible nuestro programa.

Decidimos dividir en 2 nuestra estrategia de implementación, por un lado la estructura encargada de almacenar todos los libros y por otro lado la estructura encargada de generar el índice de géneros. Esta división fue planteada pensando en: la relación que existía entre los géneros y los libros, partiendo de ello deducimos que si se buscaba directamente desde el libro el costo de búsqueda en nuestro programa iba a ser alto, entonces decidimos la creación de una estructura a partir de la otra sin la necesidad de tener que acceder a la primera para obtener la información requerida por el usuario.

**Etapa 1**

Estructura utilizada para almacenamiento de libros sin distinción en orden:

Elegimos la estructura basándonos en la funcionalidad que iba a tener en nuestra implementación, nuestra forma de resolver el problema fue, una vez obtenido el libro del archivo .csv, el mismo era agregado a la estructura sin un orden específico, ya que no era pedido en el enunciado, y automáticamente conectado a la estructura del índice, por lo que nuestro mayor problema en este punto era mejorar la eficiencia al momento de agregar cada libro y que soportara una cantidad ilimitada de libros, nos decidimos a realizar el análisis con 2 estructuras LinkedList VS ArrayList.

**LinkedList: lista doblemente enlazada**

* Acceso posicional costoso
* Inserción y extracción costosas
* **Menos en la primera y última posición que es inmediato**
* **Tamaño ilimitado**

**ArrayList: lista implementada con un array**

* Acceso posicional eficiente
* **Inserción y extracción costosas menos en la última posición que es instantánea**
* [**Cuando se supera el tamaño del array, se crea uno nuevo más grande y se copian en él los elementos del antiguo**](https://www.redeszone.net/2012/03/05/curso-de-java-estructuras-de-datos-arraylist-y-linkedlist/)**.**
* **No es óptima cuando lo que se desea manipular tiene demasiados elementos.**

Si bien en ambas estructuras plantean el costo de agregar un elemento en O (1), la realidad es que ArrayList cuenta con la desventaja marcada anteriormente una vez superado el tamaño de array se crea uno nuevo y se copian todos los elementos. Esto trae aparejado el segundo inconveniente la capacidad que tiene de soportar mucha cantidad de elementos. Por estos dos motivos decidimos usar **LinkedList.**

**Etapa 2**

Para la creación del índice, se propuso analizar dos estructuras de la interfaz List contra el árbol binario de búsqueda. Nosotros elegimos volver a evaluar el ArrayList, la LinkedList pero esta vez contra el árbol binario de búsqueda. Si tomamos en cuenta la justificación anterior de porque no elegimos ArrayList como estructura principal, en el caso del índice su función principal es la búsqueda. Asique sería una opción viable pero seguimos teniendo el problema de lo costosa que es la inserción.

Por ello encontramos que el árbol binario de búsqueda soluciona la búsqueda y la inserción. Analizándolo en el peor de los casos puede llegar a generarse formando una lista, pero existen muchas probabilidades de que esto no pase.

Siendo que las probabilidades de que se comporte como árbol es mayor a que se comporte como lista, esta estructura reúne los dos requisitos que nos parecen importantes para la implementación: la búsqueda y la inserción. Por ello lo elegimos.

Cantidad de Nodos Máximos Visitados en cada DataSet:

Este gráfico muestra probando en el método de Inserción cuantas veces como máximo visita los nodos en el árbol, en los distintos DataSet provistos por la cátedra.

Cantidad Géneros:

DataSet1: 37

DataSet2, DataSet3, DataSet4:40

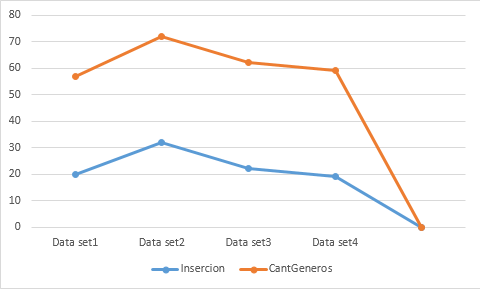
Cantidad Nodos Visitados:

DataSet1:20

DataSet2:32

DataSet3:22

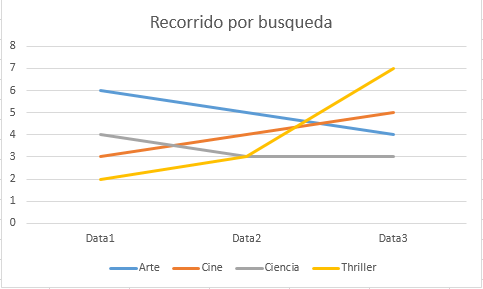
DataSet4:19



Iteraciones en la búsqueda:

En este caso tomamos cuatro géneros distintos, principalmente arte y thriller que alfabéticamente se ubican al principio y al final.

Los resultados obtenidos fueron bastante alentadores ya que de un total máximo de 40 géneros, tuvimos en el DataSet 3 un máximo de 7 iteraciones con thriller.



|  |  |  |  |
| --- | --- | --- | --- |
|  | dataset1 | dataset2 | dataset3 |
| arte | 6 | 5 | 4 |
| cine | 3 | 4 | 5 |
| ciencia | 4 | 3 | 3 |
| thriller | 2 | 3 | 7 |

**Conclusiones**

Durante el desarrollo del trabajo práctico fuimos analizando pros y contras de diferentes estructuras de almacenamiento, en base a los análisis hechos anteriormente nos decantamos por la implementación ya presentada.

Por otro lado durante la implementación, tuvimos inconvenientes con el dataset4 debido a la capacidad de memoria que tenía la máquina. Esto fue un limitante a la hora de presentar resultados. Al momento de esta entrega, sabemos que genera el árbol de géneros y lo recorre pero no termina de cargar los libros por el espacio en memoria.

De esta primera entrega nos llevamos la práctica de buscar la herramienta adecuada para la consigna dada. No descartamos otras implementaciones que quizás en algún punto pueden llegar a ser más óptimas que la propuesta por nosotros. Aclaramos esto porque al momento de hacer los análisis de eficiencia, en el árbol detectamos que es la mejor solución si tomamos una búsqueda común en un ArrayList y en una Linked, pero nos dimos cuenta que al momento de la implementación y redacción del informe no tuvimos en cuenta la eficiencia de la búsqueda binaria en un ArrayList, en este punto hubiera sido optima una implementación con esta estructura. Suponiendo que la cantidad de generos es finita y casi despreciable. Pero como no teniamos documentacion para saber apartir de cuantos indices el arraylist se tenia que multiplicar nos seguimos decantando por el arbol y su insercion. El arbol podria haber tenido la misma eficiencia de busqueda que la busqueda binaria en un array si lo hubieramos balanceado pero tambien evaluamos el costo en tiempo de codificacion que implicaba eso y que igualmente sin balanceo seguia siendo una buena estructura.

Fuentes:

<https://www.redeszone.net/2012/03/05/curso-de-java-estructuras-de-datos-arraylist-y-linkedlist/>

<http://www.enrique7mc.com/2016/07/diferencia-entre-arraylist-y-linkedlist/>

<https://code.i-harness.com/es/q/4ec9b>

https://www.aprenderaprogramar.com/index.php?option=com\_content&view=article&id=605:interface-list-clase-linkedlist-api-java-ejercicio-diferencias-entre-arraylist-y-linkedlist-codigo-cu00921c&catid=58&Itemid=180