Trabajo Práctico Especial
1ra Entrega

Programación III - TUDAI

Facultad de Cs. Exactas
Univ. del Centro de la Pcia. de Bs. As.

GRUPO: **IGARTUA, S. Lorena**
lorenaigartua@gmail.com

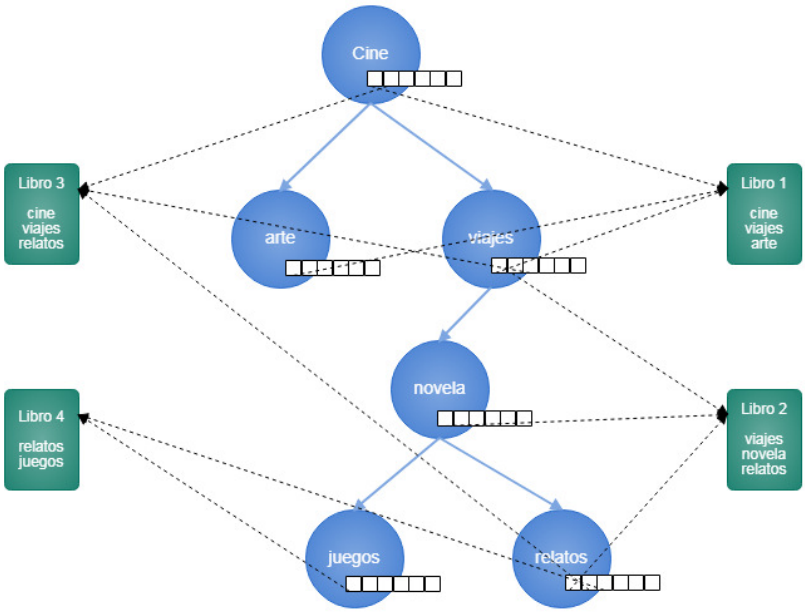
MUJICA, Martin S.
mujicamartin@gmail.com

Tandil, 04 de Mayo de 2018

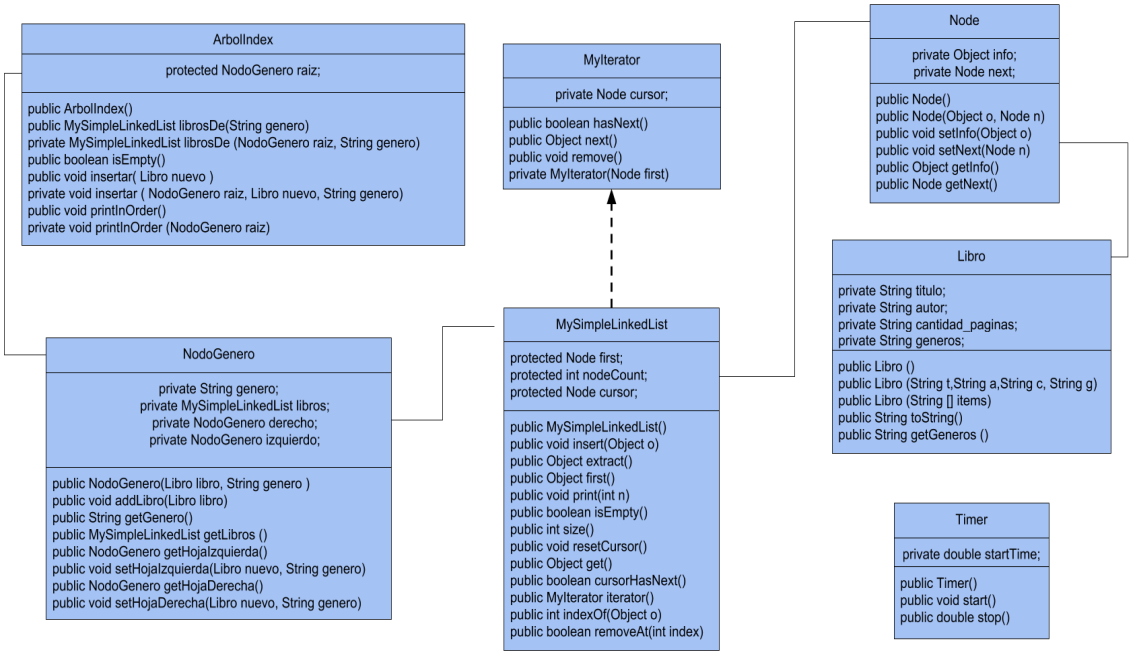
Conforme el requerimiento, partiendo de una colección de libros, se debía implementar una herramienta que permita simplificar la búsqueda de libros por géneros.

Partiendo de la lectura del archivo .csv, se generaron los objetos Libro correspondientes, cada uno con los siguientes atributos: un título, un autor, una cantidad de páginas y un conjunto de géneros, que describen el contenido del libro.

Generados los libros en memoria, y con el objeto de generar el índice por genero de los mismos, se estructuro un Arbol Binario de Búsqueda, donde cada nodo sea capaz de contener el nombre del género, y mantener una Lista Simple de punteros a aquellos libros que contengan ese género en particular.



Las clases que se usaron son las siguientes:



Como se puede observar en el gráfico, uno de los atributos de la clase Libro es géneros. Se analizaron varias alternativas, y se optó por almacenar los datos en un String (que iba a tener este formato: arte biología viajes), para recién explotar su contenido al momento de indexar el Libro por género. Esto así porque, si almacenábamos cada genero en una lista, íbamos a necesitar un primer recorrido en su armado, y un segundo recorrido, al momento de indexarlo para armar la Lista por género. Así, se ahorro espacio en memoria durante el proceso.

Otra decisión importante durante el desarrollo, fue la elección de la estructura a utilizar en el armado del índice. Para esto, se consideraron las siguientes opciones:

- Una lista simplemente vinculada: en esta estructura, cada nodo incluye un campo llamado *puntero* al siguiente elemento de la lista. Para incorporar, en cada nodo de esta lista, el genero del libro a indexar, se debería recorrer la lista para determinar la existencia del genero en cuestión, y esto implica un costo O(n). Mismo costo generaría retornar el listado de libros por género, al desconocer a que altura de la estructura esta alojado.
- ArrayList, de la interface List de Java: Internamente, el ArrayList mantiene sus contenidos usando un Array. Cuando insertemos un Nodo, el ArrayList necesitará expandir su matriz interna si se queda sin espacio. Esto podría afectar el rendimiento,

ya que siempre es mejor establecer la capacidad inicial que nuestro programa necesitará. Por otro lado, el ArrayList es bueno para la recuperación de elementos de una posición específica, con un costo en tiempo constante *O(1).* Sin embargo, agregar y eliminar elementos, de una manera desordenada, tiene un costo lineal *O (n).*

- Un árbol binario de búsqueda: La idea básica de este tipo de estructura de datos es crear un árbol en el cual cada vértice tiene a lo más dos hijos (comúnmente llamados izquierdo y derecho). Los vértices tienen la propiedad de que todos los datos de la ramificación izquierda son menores al del vértice, mientras que los de la derecha son mayores. Esto implica que, al momento de consultar los libros asociados a un genero en particular, el costo es O(log 2 n).

Con este contexto, viendo que una de las principales desventajas de las listas enlazadas es que si queremos buscar un dato, tenemos que recorrer todo el arreglo y que, para la interface List, desconocíamos la cantidad de géneros, optamos por utilizar el árbol binario de búsqueda.

Dentro del ArbolIndex diagramado, y tal como se observa en el gráfico, cada NodoGenero contiene el nombre del género, y una Lista Simple de Libros que contiene el genero en cuestión. Se selecciono esta estructura, por la ventaja que tiene por sobre un ArrayList u otra interface de List, en cuanto al modo de expandirse. Así, cada Lista tendrá tantos nodos como Libros incorporados, sin espacios libres. Por otro lado, como la aplicación no necesita recuperar un libro en particular, sino todos los libros del género, no se justificaba la utilización de una estructura tipo árbol.

A fin de cumplimentar la Etapa 1 del requerimiento, se implementaron los métodos para obtener la colección de libros que contenga un género en particular, ingresado por el usuario. El programa genera un archivo .csv con los títulos de los libros que cumplen con el género dado.

A su vez, se diseñó un índice alfabético, que indica la totalidad de géneros disponibles, a fin de simplificar el acceso a solo un subconjunto de todos los libros existentes. Para esto, optamos por acceder a los generos recorriendo el Árbol Binario de Busqueda Index, en un recorrido InOrder, que implica realizar las siguientes operaciones recursivamente en cada nodo:

- atravesar el sub-árbol izquierdo
- Visitar la raíz
- atravesar el sub-árbol derecho

De esta manera, el índice de géneros que se obtiene estará ordenado alfabéticamente, facilitando su uso por el usuario.

A fin de probar las selecciones, se implementó la solución usando ArrayList en reemplazo de ListaSimple dentro del Nodo Genero, y asignando al Libro como atributo, un ArrayList de generos. Esta solución, implica recorrer el String de generos de cada Libro, para el armado del ArrayList, y que el árbol, para armar el índice, vuelva a recorrerlo para evaluar cada genero y resolver su inclusión en cada NodoGenero y su ArrayList asociado. Estos recorridos, conllevan uso de memoria, y, por tanto, demora. Usando la clase Timer(), obtuvimos los siguientes datos:

Archivo	generación e indexación de libros		Consulta de un género (y archivo salida)	
	Solucion con ListaSimple	Solucion con ArrayList	Solucion con ListaSimple	Solucion con ArrayList
dataset1.csv	7.490771 nanosegundos	6.436362 nanosegundos	5.629619 nanosegundos	3.535457 nanosegundos
dataset2.csv	38.287711 nanosegundos	40.783413 nanosegundos	10.050954 nanosegundos	11.221869 nanosegundos
dataset3.csv	626.860969 nanosegundos	1243.284098 nanosegundos	1187.830556 nanosegundos	102.110576 nanosegundos
dataset4.csv	10469.850067 nanosegundos	17629.600673 nanosegundos	6864.567472 nanosegundos	9196.739173 nanosegundos

La estructura seleccionada condiciona el uso de memoria y el costo de demora de cada método que implementemos. Esto será mas palpable, cuantos más datos debamos procesar.