Trabajo práctico especial.

Entrega Nº 1.

Programación 3 - TUDAI 2022
UNICEN

Fecha de entrega 01/07/2022

Alumno: Farinelli, Matías E.
matfarinelli@gmail.com
https://github.com/matfarinelli/TP-Especial-Prog-III

Contenido

Introducción:

A lo largo del presente informe se pretende desarrollar la solución propuesta por el alumno y su implementación, a una problemática planteada por la cátedra. Se analizará cada servicio solicitado, buscando la optimización computacional aplicando los conocimientos vistos durante la cursada.

Problemática planteada:

Partiendo de una colección de libros, se desea implementar una herramienta que permita simplificar la búsqueda de libros por géneros; además de caracterizar el comportamiento de los usuarios mientras realizan dichas búsquedas.

Cada libro se compone de un título, un autor, una cantidad de páginas y un conjunto de géneros, que describen el contenido del libro. Ejemplos de estos géneros son arte, ciencia, policial, entre otras.

En esta primera etapa se desea implementar la lógica necesaria 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.

La cátedra provee de un conjunto de datasets, de diferentes tamaños, donde cada línea del documento representa un libro, su autor, y los géneros del mismo.

También se provee del algoritmo a utilizar, para mediante un archivo denominado “CSVReader”, se pueda fraccionar e inspeccionar línea a línea del dataset para la generación de los libros. Además de lo mencionado, se brinda un segundo archivo, “CSVWritter” el cual permite generar un archivo .csv con los resultados obtenidos por el algoritmo solución.

Desarrollo:

Empezaremos por indicar cuales fueron las clases utilizadas y su función principal en la implementación del algoritmo solución para luego abordarlos con mayor detalle.

1- “CSVReader”, algoritmo con estructura base provisto por la cátedra, que permitió leer el dataset recibido por constructor. Sus principales funciones son la instanciación del “Libro”, mediante el fraccionamiento por línea del dataset a través el método “split” y el cargado de estos mismos libros a la “Biblioteca”.

2- “Biblioteca”, es la encargada de guardar los “Libros” en un arrayList de Libros. Tiene un segundo arrayList de “Genero” donde conoce todos los géneros disponibles en la biblioteca. El método principal es “cargarBiblioteca”, el cual invoca a CSVReader.

3. “Libro”, mediante diferentes métodos se puede setear o conocer sus atributos. Principalmente invocado en CSVReader, donde estos se crean luego del fraccionamiento del .csv. Es el resultado de la búsqueda del algoritmo propuesto, por eso propone métodos de consulta minuciosa. El índice consulta “géneros” pero devuelve “Libros”.

4. “Genero”, en la implementación del índice es la clase más importante, dado que el género es quien tiene como atributo un arrayList con los libros que pertenecen a el mismo. Mediante el índice se busca un “género”, si ese género existe, solo debe indicar que libros le forman parte.

5. “Main”, para crear instancias de las clases mencionadas e invocar métodos que permitan analizar el funcionamiento del algoritmo y tomar métricas de rendimiento a los fines del presente informe.

6. “CSVWritter”, algoritmo proporcionado por la cátedra que permite generar un .csv con los resultados obtenidos por la consulta al índice desarrollado.

Implementación Índice:

Al ejecutar el método “cargarBiblioteca” ocurren dos cosas muy importantes para el desarrollo de la solución. Mediante el análisis de cada línea del dataset, instancia un “Libro” y lo añade a la Biblioteca.

```
while ((line = br.readLine()) != null) {
    // fraccionado del csv
    String[] items = line.split(cvsSplitBy);
    String genero = items[3];
    String[] generos = genero.split(" ");

    // libro(titulo,autor,paginas)
    Libro libro = new Libro(items[0], items[1], items[2]);

    // recorrido de cada genero del libro
    for (String generoLibro : generos) {
        // para omitir primera linea - titulos de csv
        if (generoLibro.equals("Generos")) {
            continue;
        }
        libro.agregarGenero(generoLibro);
    }

    biblioteca.addLibro(libro);
}
```

Con la instancia de “Libro” se guarda cada género en el arrayList<Genero> de la clase “Libro”. Su complejidad es O(n) por el método *contains*.

Se escogió la estructura del arrayList, dado que para la solución planteada no se necesita que los géneros de la clase Libro estén ordenados y podremos consultar los géneros de un libro con complejidad O(1). Si se hubiese priorizado la inserción ordenada, hubiésemos obtenido un mayor costo computacional en el proceso de carga y que no fue el utilizado a la hora de implementar el índice.

```java
public class Libro {

    private String titulo;
    private String autor;
    private String cantidadPaginas;
    private ArrayList<String> generos;
```

Método “addLibro” de la clase Biblioteca:

La función de este método es añadir el Libro recibido a la biblioteca, la cual posee en sus atributos un arrayList de Libros y otro de Géneros.

```java
public class Biblioteca {

    private ArrayList<Libro> libros;
    private ArrayList<Genero> generos;
```

Desglosando el método, se puede observar que al finalizar, el libro es añadido al arrayList<Libro> de la biblioteca con una complejidad O(1), dado que la inserción es simple, sin ordenamiento.

La complejidad del método está dada por la inserción del libro en el arrayList<Libro> perteneciente a la clase Género y por la inserción ordenada de los géneros en el arrayList<Genero> de la clase Biblioteca. **Para la implementación del índice, es el Género quien conoce los libros que son propios o pertenecen a su género.**

Biblioteca recibe un nuevo libro, y analiza uno a uno los géneros de ese libro y los compara con los géneros que biblioteca tiene en su arrayList<Genero>.

Puede ocurrir que el género ya esté en su arrayList, en ese caso el Género (existente) añade el libro a su índice, o puede ocurrir que no exista, lo crea (instancia) y el Género (nuevo) añade el libro a su índice. De dicha operación se desprende una complejidad O(n^2), dado que debe recorrer todos los géneros del libro recibido para añadirlo en cada índice y en el peor de los casos habrá recorrido todos los géneros arrayList<Genero> de Biblioteca. N es cantidad de géneros.

```java
public void addLibro(Libro libro) { //O(n^2)
    // generos del nuevo libro
    ArrayList<String> generosLibro = libro.getGeneros();

    for (String generoLibro : generosLibro) { //O(n)
        // obtengo el primer genero y lo voy comparando con los generos de la biblioteca
        Boolean esta = false;

        for (Genero generoBiblioteca : this.generos) { // O(n)

            if (generoBiblioteca.getNombre().equals(generoLibro)) {
                // al indice
                generoBiblioteca.addLibro(libro); //O(1)
                esta = true;
            }
        }

        // si no esta el genero, agregarlo a biblioteca
        // y agregar el libro a su organizador en "generos"
        if (!esta) {
            Genero nuevoGenero = new Genero(generoLibro);
            this.addGenero(nuevoGenero);
            // metodo que inserta genero ordenado a la biblioteca
            nuevoGenero.addLibro(libro);
        }

    }

    this.libros.add(libro);
}
```

Otro método que aporta complejidad es el método addGenero, quien añade el género de manera ordenada en el arrayList<Genero> de la clase Biblioteca.
Primero, chequea que el género recibido no esté en el arrayList de la biblioteca lo cual nos da un O(n) propio de *contains*. En caso que no esté, el algoritmo busca la posición del nuevo género para que la inserción sea ordenada mediante un while O(n). La complejidad resultante es exponencial O(n^2).

```java
public void addGenero(Genero genero) { // O(n^2)
    if (this.generos.isEmpty()) {
        this.generos.add(genero);
    } else {
        if (!generos.contains(genero)) { //O(n)
            int posicion = 0;

            // 0 es igual - 1 es mayor y -1 es menor
            while (posicion < generos.size() && (generos.get(posicion).compareTo(genero.toString()) < 0)) {
                posicion++;
            }

            if (posicion == generos.size()) {
                // agrego al final
                generos.add(genero);
            } else {
                this.generos.add(posicion, genero);
            }

        }
    }

}
```

Desarrollo índice:

Como solución a la problemática propuesta y con las estructuras disponibles para esta primera entrega, el índice se implementó sobre arrayList priorizando la complejidad constante para retornar consultas O(1). Los géneros son alojados en una arrayList<Genero> en Biblioteca, y mediante una consulta a dicha lista, podemos consultar el arrayList<Libro> de dicho género, ambas en tiempos constantes.

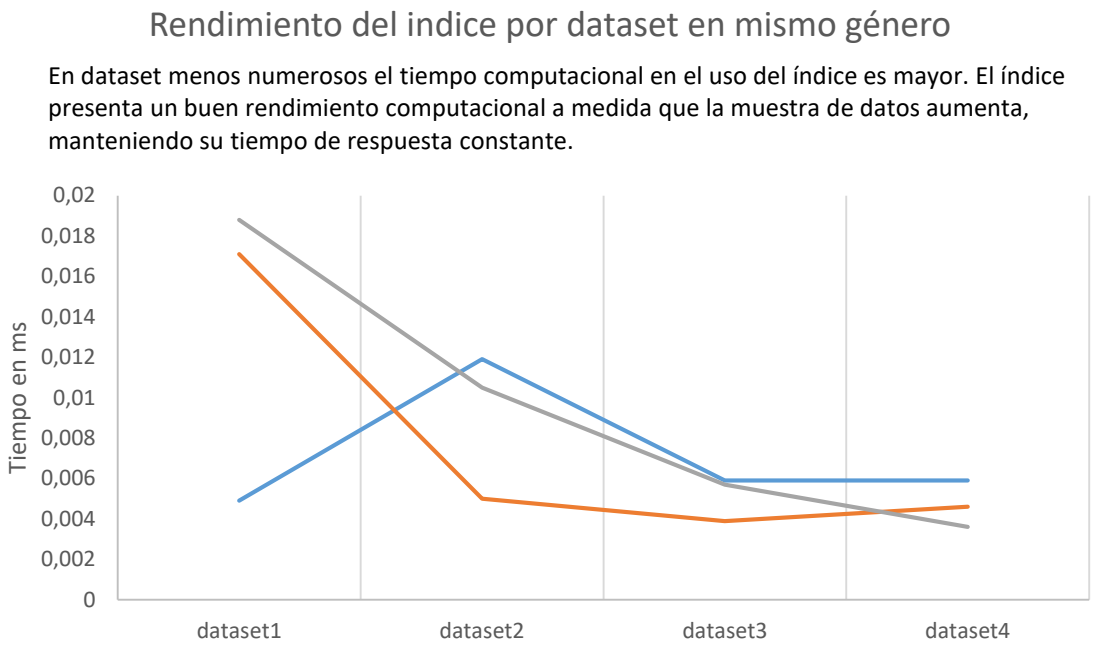
```java
// llamo al indice del genero
public ArrayList<Libro> getLibrosPorGenero(String genero) {
    Genero generoBuscado = new Genero(genero);

    Timer timer = new Timer();
    timer.start();

    for (Genero generoBiblioteca : this.generos) {

        if (generoBiblioteca.equals(generoBuscado)) {
            System.out.println(timer.stop());
            return new ArrayList<>(generoBiblioteca.getLibros());
        } // else
          // System.out.println("no tenemos el genero buscado");
    }

    System.out.println("Busqueda por índice en biblioteca: ");
    System.out.println(timer.stop());
    return new ArrayList<Libro>();

}
```

Se expresan las métricas y tiempos computacionales del algoritmo implementado en razón de los diferentes datasets utilizados.

Se ejecutó 3 veces la misma consulta al índice con cada dataset. Los resultados arrojaron que el tiempo para retornar el conjunto solución (o arrayList) obtenido por el índice no es proporcional al tamaño del arraylist retornado ni la muestra analizada. Al contrario, los resultados indican que a menor cantidad de datos el dataset, el tiempo de procesamiento para consultar el índice y retornar el conjunto de libros, es

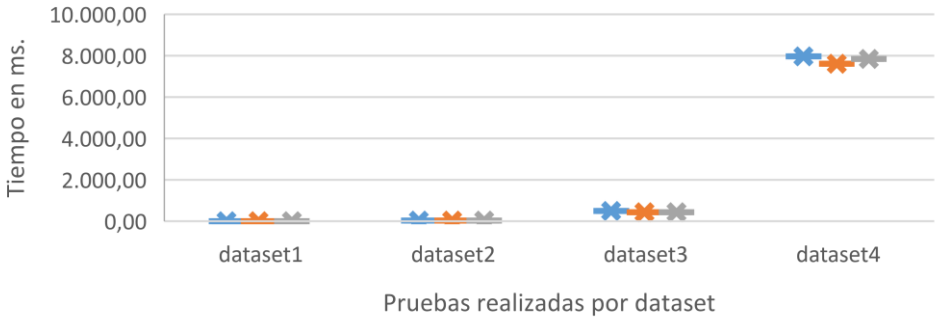
mayor. El índice presenta buen rendimiento computacional, a medida que el conjunto de datos indexados, crece.

	dataset1	dataset2	dataset3	dataset4
ArrayList<Libro> getLibrosPorGenero("humor");	0,0049	0,0119	0,0059	0,0059
ArrayList<Libro> getLibrosPorGenero("humor");	0,0171	0,005001	0,003899	0,004601
ArrayList<Libro> getLibrosPorGenero("humor");	0,0188	0,0105	0,005701	0,0036



No ocurre lo mismo con la carga de datos, a mayor tamaño el dataset, mayor tiempo de procesamiento. Los motivos fueron explicados en la implementación y sus costos computacionales.

	dataset1	dataset2	dataset3	dataset4
tiempo de carga biblioteca	4,42	27,25	491,08	7.968,99
tiempo de carga biblioteca	5,11	31,41	426,00	7.604,80
tiempo de carga biblioteca	4,98	35,00	435,61	7.836,11



Conclusión:

Mediante las pruebas realizadas sobre el índice, pudo verificarse el buen desempeño de la estructura elegida para guardar los datos y expresar sus resultados. Se analizaron datasets con 20, 1.000, 100.000 y 1.000.000 de libros. El índice mantuvo la constancia y generó resultados en un tiempo uniforme a lo largo de las muestras, indicando que la mayor parte del tiempo computacional se insume en la generación de la biblioteca y en el guardado de los datos en cada estructura.

Conceptualmente, la implementación de un árbol binario en la estructura del índice, podría haber sido el óptimo en materia de gasto computacional. La inserción ordenada de cada género en los nodos, brindaría un tiempo O(log n) en la búsqueda y generación de los resultados. Aun así, la estructura utilizada cumple con el objetivo y su desempeño ante las pruebas realizadas, es aceptable.