75.06 Organización de Datos

TP N°1: Catálogo discográfico

Grupo 07

"Documentación de diseño"

Índice

1. Introducción

El siguiente documento tiene como objetivo especificar y precisar los detalles técnicos de la realización de la primera parte del trabajo práctico. Se explican los problemas surgidos a la hora del diseño y las soluciones encaradas para solventar los mismos.

Todos los archivos y códigos fuente aquí mencionados y relacionados al proyecto, así como también el presente informe, pueden ser encontrados y descargados del repositorio del grupo (`https://github.com/federicomrossi/7506-tp-grupo07`).

2. Requerimientos planteados

El trabajo solicitado se basa en la creación de un sistema para la generación de un catalogo a partir de archivos de letras de canciones de diversos autores. Dicho sistema debe ser capaz de realizar búsquedas por autor, titulo y frases, entregando como resultado los temas que coincidan con los parámetros ingresados.

A su vez se exigieron ciertas condiciones para la realización del trabajo a la hora del manejo de las estructuras. El índice para la búsqueda por titulo debía estructurarse como un hash, por autor como un árbol B+ y los archivos de texto como bloques de registros variables.

El producto final es una aplicación de consola que permite realizar las operaciones pedidas.

3. División del trabajo

El trabajo esta divido en tres capas funcionales: la física, la lógica y el front-end.

A su vez, la capa lógica esta formada por tres módulos principales, a saber, el árbol B+, el hash extensible y las estructuras de recuperación de textos. La distribución mencionada se puede observar en la *Figura 1.*

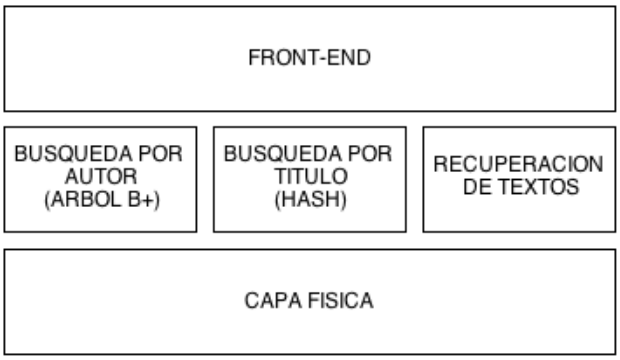

Figura 1: *Esquema de la estructuración en capas.*

Esta división esta fundamentada en tener el menor acoplamiento posible entre las clases y sobretodo entre los módulos lógicos. A su vez, permite una fácil división de las tareas entre los integrantes del grupo de trabajo.

4. Detalles técnicos

A continuación se describirá con un mayor nivel de detalle cada una de los módulos que componen el sistema.

4.1. Capa Física

La capa física de este trabajo se centra alrededor de dos clases: *ArchivoBloques* y *SerialBuffer*.

La primera es la encargada de la interacción directa con el disco. La misma define interfaces para poder leer y escribir un archivo en función de bloques de un tamaño fijo y parametrizable.

El problema que surgió aquí es que independientemente de utilizar el modo normal de apertura para escribir de C++ (ios::out), el archivo era truncado y se perdía su contenido, quedando solamente el último bloque escrito. La solución para esto fue que si se quería modificar un bloque ya existente, se utilizara un archivo de trabajo temporal para pasar el contenido original, intercalar el bloque a modificar y luego copiar el resto del archivo. Posteriormente se elimina el archivo original y se renombra el de trabajo. Debido a que esto es muy costoso y generalmente muchas de las operaciones de escritura consisten en agregar un nuevo bloque al final, existe un método que abre el archivo en modo append (ios::app) y agrega el nuevo bloque. La lectura en cambio no presento problema alguno.

SerialBuffer es la clase encargada de brindar los medios para poder persistir de manera ordenada los registros de cada estructura. La misma presenta dos métodos principales, pack y unpack.

El primero se encarga de agregar registros a un buffer de caracteres (cabe aclarar que se escogió por archivos de tipo binario para la persistencia de datos). Para poder recuperar la información a posteriori, antes de empaquetar cada registro carga en el buffer un prefijo de longitud para poder saber cuanto va a tener que recuperar del buffer a un objeto.

El segundo, hace el trabajo inverso y restaura la información de un buffer, que previamente se leyó desde el disco, a un objeto. Utiliza el prefijo de longitu para saber la cantidad de caracteres a pasar desde el buffer al objeto de destino.

Como puede notarse, el buffer es la estructura fundamental y puede verse al mismo como una sucesión de registros como la siguiente:

```
[prefijoLongitud(unsigned short int), reg(registro genérico de longitud variable)]
```

Donde el tamaño total queda comprendido dentro del tamaño del buffer.

Una carácteristica de esta implementación es la relación uno a uno que debe haber entre los tamaños de bloque y de los buffer, ya que si los mismos no coincidieran generarían errores de segmentación a la hora de tratar datos en memoria.

4.2. Arbol B+

[COLOCAR TEXTO AQUI]

4.3. Hash Extensible

[COLOCAR TEXTO AQUI]

4.4. Front-end (Interfaz)

[COLOCAR TEXTO AQUI]

4.5. Conclusiones

[COLOCAR TEXTO AQUI]