Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI2692 - Laboratorio de Algoritmos y Estructuras de Datos II
Trimestre Enero-Marzo 2024

Administrador de Música

El objetivo del proyecto es la implementación de una aplicación en Kotlin para administrar y reproducir archivos de audio en formato MP3. Se requerirá que implemente cuatro módulos, tres de ellos consisten en tipos abstractos de datos (TADs) y uno integra los TADs para hacer una aplicación que sirva para administrar y reproducir música.

1. Tipos de datos la aplicación

A continuación se presentarán los tipos de datos que componen el Administrador de música.

1.1. TAD Canción

El TAD Canción tiene como objetivo representar la información relacionada con un archivo de audio que se asume contiene una canción. Para cada canción se debe tener su *título*, su *intérprete*, y la *ubicación* del archivo de audio con la música. La *ubicación* debe corresponder a un dirección absoluta desde la raíz del sistema de archivos hasta el archivo de audio. Teniendo como restricción que el sistema de archivo es el que usa el sistema de operación Linux. Por ejemplo, si un archivo de música se llama `‘‘Beethoven.9na-Sinfonia.mp3’’`, su título podría ser `‘‘9na Sinfonia de Beethoven’’`, el intérprete es la `‘‘Filarmónica de Berlín’’` y su ubicación es `‘‘/home/gpalma/MiMusica/Beethoven.9na-Sinfonia.mp3’’`. Se define la función `esUbicacionValida`, como la función que recibe como argumento como un String que corresponde a la dirección absoluta a un archivo en formato MP3, y retorna `True` en caso de que el archivo exista, y `False` en caso contrario. La Figura 1 presenta la especificación del TAD Canción.

1.2. TAD Reproductor

El TAD Reproductor es el encargado de proporcionar las facilidades para la reproducción de canciones. La Figura 2 muestra el TAD Reproductor. La idea es hacer transparente a un usuario, las operaciones de un reproductor de música. Para poder lograr este objetivo el módulo debe hacer uso de alguna librería externa que permita la reproducción de archivos de sonidos, en específico, los archivos de sonido de formato MP3, los cuales son los tipos de archivo que maneja los objetos de tipo Canción. El reproductor debe tocar una canción, debido a esto en el modelo de representación el reproductor contiene a un objeto de tipo `Canción` que va a ser la canción cargada en reproductor para su reproducción. Las operaciones de este TAD son las comunes de un reproductor de música. El método `cargarCancion` permite a un reproductor cargar una nueva canción a reproducir. El método `estaTocandoCancion` detecta si en el momento actual la librería de audio está reproduciendo un archivo de sonido o no.

1.3. Tipo de dato Árbol de Canciones (`ArbolDeCanciones`)

El tipo de dato `ArbolDeCanciones` es un *árbol binario de búsqueda* con las siguientes características:

(I) Se define un `ArbolDeCanciones` vacío como aquel que tiene la raíz igual al elemento `NULL`. (II) Cada nodo contiene es elemento del tipo `Canción`, que lo denominamos c. (III) También los nodos tienen apuntadores al nodo padre, al hijo izquierdo y al hijo derecho. (IV) Sea A un nodo, y sea *HijoDer* un nodo que es el hijo derecho de A, entonces se tiene que cumplir que $(A.c.interprete < HijoDer.c.interprete) \vee ((A.c.interprete = HijoDer.c.interprete) \wedge (A.c.titulo < HijoDer.c.titulo))$. (V) Sea A un nodo, y sea *HijoIzq* un nodo que es el hijo izquierdo de A, entonces se tiene que cumplir que $(A.c.interprete > HijoIzq.c.interprete) \vee ((A.c.interprete = HijoIzq.c.interprete) \wedge (A.c.titulo > HijoIzq.c.titulo))$. (VI) Un nodo hoja, es aquel que no tiene ni hijo izquierdo, ni derecho. (VII) En el árbol no se acepta que dos nodos diferentes contengan la misma canción. Es decir, las canciones son únicas en el árbol.

Especificación del TAD Canción

Modelo de Representación

const *titulo* : String .
const *interprete* : String .
const *ubicacion* : String .

Invariante de Representación

$titulo \neq \text{NULL} \ \wedge \ interprete \neq \text{NULL} \ \wedge \ ubicacion \neq \text{NULL} \ \wedge \ esUbicacionValida(ubicacion)$

Operaciones

constructor *crearCancion* (**in** t : String; **in** i : String; **in** u : String) → Canción
{ **Pre:** $t \neq \text{NULL} \wedge i \neq \text{NULL} \wedge u \neq \text{NULL} \wedge esUbicacionValida(u)$ }
{ **Post:** $self.titulo = t \ \wedge \ self.interprete = i \ \wedge \ self.ubicacion = u$ }

fmeth *obtenerTitulo* ( ) → String
{ **Pre:** True }
{ **Post:** $obtenerTitulo = self.titulo$ }

fmeth *obtenerInterprete* ( ) → String
{ **Pre:** True }
{ **Post:** $obtenerInterprete = self.interprete$ }

fmeth *obtenerUbicacion* ( ) → String
{ **Pre:** True }
{ **Post:** $obtenerUbicacion = self.ubicacion$ }

fmeth *toString* ( ) → String
{ **Pre:** True }
{ **Post:** *toString* = String que muestra el $self.titulo$ y el $self.interprete$ de la canción }

Fin TAD

Figura 1: Especificación del TAD Canción.

Especificación del TAD Reproductor

Modelo de Representación

var *actual* : Canción .

Invariante de Representación

actual $\neq$ NULL .

Operaciones

constructor *crearReproductor* (**in** c : Canción) $\rightarrow$ Reproductor
{ **Pre:** True }
{ **Post:** $self.actual = c$ }

meth *cargarCancion* (**in** c : Canción)
{ **Pre:** True }
{ **Post:** $self.actual = c \wedge$ Se carga en la librería de audio el archivo $c.ubicacion$ }

meth *reproducir* ( )
{ **Pre:** True }
{ **Post:** Se reproduce el archivo de audio $self.actual.ubicacion$ }

meth *parar* ( )
{ **Pre:** True }
{ **Post:** Se detiene la reproducción del archivo de audio $self.actual.ubicacion$ }

meth *pausa* ( )
{ **Pre:** True }
{ **Post:** Se pausa la reproducción del archivo de audio $self.actual.ubicacion$ }

fmeth *estaTocandoCancion* ( ) $\rightarrow$ Boolean
{ **Pre:** True }
{ **Post:** *estaTocandoCancion* $\equiv$ **Si** el archivo de audio $self.actual.ubicacion$
se está reproduciendo **entonces** True **de lo contrario** False }

Fin TAD

Figura 2: Especificación del TAD Reproductor.

Observe que para ordenamiento de los objetos en el árbol se hace uso de dos claves. La clave primaria es el intérprete de la canción y la secundaria es el título de la canción. Es decir, si hay dos nodos con canciones que poseen intérpretes diferentes, entonces el intérprete es la clave usada para el ordenamiento. Si por el contrario dos canciones tienen el mismo intérprete, entonces se usa el título para determinar la posición de la canción en el árbol.

1.4. TAD Lista de Reproducción (LR)

Un administrador de música no está completo sin una lista de canciones a reproducir. El objetivo es tener una estructura dinámica que contenga las canciones que van a ser reproducidas. La estructura a utilizar para almacenar las canciones debe ser un `ArbolDeCanciones`. En específico se usará en el modelo de representación del TAD LR un `ArbolDeCanciones`, en los que cada nodo contiene un elemento de tipo `Canción`. En el invariante de representación se exige que la variable de tipo `ArbolDeCanciones` sea un árbol binario de búsqueda y que todos los elementos de tipo canción, que se encuentran almacenados en los nodos, sigan el orden deseado usando la claves intérprete y título. Para verificar estas condiciones se define la función `esArbolDeBusqCancion` inductivamente sobre la estructura `ArbolDeCanciones` como muestra la Figura 3. En la función `ArbolDeCanciones` se tiene que c, que es un elemento de tipo `Canción`. Las funciones `minInterprete` y `maxInterprete` determinan el menor y mayor valor del String que representa a un intérprete que se encuentran en el árbol de búsqueda binaria, usando ordenamiento lexicográfico. De la misma manera las funciones `minTitulo` y `maxTitulo` tienen como fin encontrar el menor y mayor valor de un título de una canción del árbol binario de búsqueda.

$$
\begin{array}{l}
esArbolDeBusqCancion : ArbolDeCanciones \rightarrow \text{Boolean} \\
\quad \text{con cláusulas} \\
\quad \left[\begin{array}{l}
esArbolDeBusqCancion(\underline{\text{avac}}) \;=\; \text{True} \;\;, \\
esArbolDeBusqCancion(\underline{\text{nodo}}(izq, c, der)) = (c.interprete < minInterprete(der) \vee \\
\qquad\qquad\qquad\qquad\qquad\qquad (c.interprete = minInterprete(der) \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad c.titulo < minTitulo(der))) \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ esArbolDeBusqCancion(der) \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ (c.interprete > maxInterprete(izq) \vee \\
\qquad\qquad\qquad\qquad\qquad\qquad (c.interprete = maxInterprete(izq) \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad c.titulo > maxTitulo(izq))) \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ esArbolDeBusqCancion(izq)
\end{array} \right.
\end{array}
$$

Figura 3: Definición de la función *esArbolDeBusqCancion*. Se tiene que `avac` es un `ArbolDeCanciones` vacío, c es un elemento de tipo `Canción`, y *izq* y *der* son los subárboles izquierdo y derecho de un árbol. Las funciones `minInterprete` y `maxInterprete` encuentran el mínimo y máximo intérprete de un árbol, usando orden lexicográfico. Las funciones `minTitulo` y `maxTitulo` encuentran el mínimo y máximo título de un árbol, usando orden lexicográfico.

En la Figura 4 se muestra la especificación del TAD Lista de Reproducción (LR). La primera operación del TAD LR es `agregarLista`, que recibe como entrada el nombre de un archivo con una lista de canciones, las cuales carga en la lista de reproducción. El formato del archivo con la lista de canciones es el siguiente. Cada línea corresponde a una canción y la misma posee tres campos, el primero es el intérprete de la canción, el segundo es el título de la canción y el tercero es la dirección absoluta, en el sistema de archivos, del archivo de audio con formato MP3. Los campos están separados por ";". La Figura 5 muestra un ejemplo de un archivo con un formato válido, con una lista de cuatro canciones. Por cada línea del archivo con la lista de canciones, el método `agregarLista` crea un elemento de tipo `Canción` para luego agregarlo en la estructura árbol de canciones. La segunda operación del TAD LR es `eliminarCancion`, que remueve una canción del árbol de canciones, dado el intérprete y el título de la canción. La tercera operación es `obtenerLR` que retorna una secuencia con todas las canciones contenidas en el árbol binario de búsqueda de canciones. Para construir la secuencia de elementos de tipo `Canción` debe usar la función `deArbolASecuencia`, la cual se define inductivamente como se muestra en la Figura 6. La cuarta y última operación del TAD LR es `mostrarLR`, que muestra por la salida estándar cada canción c contenida en el árbol de canciones. El árbol se debe recorrerse *in-orden* y cada elemento c de tipo `Canción` debe mostrar su contenido haciendo aplicando método $c.aString()$.

Especificación del TAD LR

Modelo de Representación

var *contenido* : ArbolDeCanciones .

Invariante de Representación

esArbolDeBusqCancion(*contenido*).

Operaciones

constructor *crearLR* ( ) → LR
{ **Pre:** True }
{ **Post:** *self*.*contenido* = NULL }

meth *agregarLista* (**in** *na* : String)
{ **Pre:** *na* es el nombre de un archivo que posee un formato válido con una lista de canciones }
{ **Post:** Con cada linea de *na* se crea un elemento de tipo Canción
el cual es agregado a *self*.*contenido* en el orden en que aparece en *na* }

meth *eliminarCancion* (**in** *i* : String; **in** *t* : String)
{ **Pre:** True }
{ **Post:** Elimina de *self*.*contenido* la canción que tenga como intérprete a *i* y como título a *t* }

fmeth *obtenerLR* ( ) → seq
{ **Pre:** True }
{ **Post:** *obtenerLR* = *deArbolASecuencia*(*contenido*) }

meth *mostrarLR* ( )
{ **Pre:** True }
{ **Post:** Para cada canción *c* en *self*.*contenido*, se muestra por la salida estándar
el String *c*.*aString*(), recorriendo el árbol de búsqueda *self*.*contenido* in-orden }

Fin TAD

Figura 4: Especificación del TAD Lista de Reproducción.

```
Katy Perry;Dark Horse;/home/gpalma/MiMusica/katy_perry-dark_horse.mp3
Pharrell Williams;Happy;/home/gpalma/MiMusica/Pharrell HAPPY.mp3
OneRepublic;Counting Stars;/home/gpalma/MiMusica/Counting Stars.mp3
Katy Perry;Roar;/home/gpalma/MiMusica/Katy Perry - Roar.mp3
```

Figura 5: Ejemplo de un archivo que contiene la información de cuatro canciones.

deArbolASecuencia : ArbolDeCanciones → seq
con cláusulas

$$\left[\begin{array}{l} deArbolASecuencia(\underline{\text{avac}}) = [\,] \;\;, \\ deArbolASecuencia(\underline{\text{nodo}}(izq, c, der)) = deArbolASecuencia(izq) + [c] + deArbolASecuencia(der) \end{array}\right.$$

Figura 6: Definición de la función `deArbolASecuencia`. Se tiene [ ] denota una secuencia vacía. El operador + corresponde al operador concatenación de secuencias.

2. Módulo de administración de música

Una vez definidos los TADs podemos crear una aplicación que administre archivos de música. En este caso vamos a describir un módulo cliente que hace uso de los TADs como librerías. A esta aplicación la vamos a llamar *Administrador de Música* (ADM). El ADM interactúa con los usuarios por medio de menú iterativo que presenta las siguientes opciones:

1. Cargar lista de reproducción.
2. Mostrar lista de reproducción.
3. Eliminar canción.
4. Reproducir.
5. Pausar.
6. Parar.
7. Próxima canción.
8. Salir del administrador de música.

Suponemos que las entradas y salidas de las operaciones del ADM, se realizan por medio de la entrada y salida estándar. A continuación se explican cada una de las opciones.

Cargar lista de reproducción: Solicita al usuario el nombre del archivo de datos con la lista de canciones. El archivo debe estar en el formato válido explicado anteriormente. Se debe crear un objeto que corresponda a la `Lista de Reproducción` y almacenar en él las canciones especificadas en el archivo. Un usuario puede usar esta opción para cargar varios archivos de datos de canciones y de esa manera aumentar el número de canciones de la lista de reproducción.

Mostrar lista de reproducción: Muestra por la salida estándar las canciones almacenadas hasta ahora en la `Lista de Reproducción` del administrador de música.

Eliminar canción: Debe mostrar la lista de reproducción y permitir al usuario seleccionar la canción que quiere remover y luego esta es eliminada de la lista de reproducción.

Reproducir: Antes de reproducir cualquier archivo de audio es necesario que se hayan cumplido varios pasos. Se debe haber cargado la librería de audio y se debe tener la lista de las canciones a reproducir. La lista de las canciones a reproducir se obtiene mediante la ejecución del método `obtenerLR` de un objeto `LR`. Una vez que se empieza a reproducir un archivo MP3, se debe mostrar al usuario el nombre del intérprete y el título de la canción que se está reproduciendo.

Pausa: Hace una pausa en la reproducción del archivo MP3. Se debe mostrar al usuario el nombre del intérprete y el título de la canción que esta en pausa. Si después de pausar el usuario manda a *reproducir* la canción, entonces la canción se debe reproducir desde el momento en que se pausó.

Parar: Si en el momento de escoger esta opción hay una canción reproduciéndose, entonces se debe parar su reproducción. En la lista de reproducción esta canción que se paró es la próxima a tocarse. Si luego de parar una canción, el usuario selecciona la opción de **Reproducir**, entonces la canción comienza desde el principio.

Próxima canción: Al seleccionar esta opción se debe comenzar a reproducir la próxima canción en la lista de reproducción, ya sea que en ese momento se este tocando una canción o no. En caso de que haya una canción reproduciéndose, entonces de debe parar su reproducción y se procede a reproducir la próxima en la lista de reproducción. Igual que para reproducir, se debe mostrar al usuario el nombre del intérprete y el título de la canción que se esta reproduciendo.

Salir del administrador de música: Se termina la ejecución de la aplicación.

Es responsabilidad de este módulo verificar las precondiciones antes de llamar a las métodos de los TADs, en este caso se le debe indicar al usuario que la operación no pudo ser efectuada porque no se cumple la precondición, indicando cual es esa precondición.

3. Requerimientos de la implementación

Cada uno de los TADs y el tipo de datos `ArbolDeCanciones`, deben ser implementado como una clase de `Kotlin`. El programa cliente ADM debe ser implementado en un archivo aparte. Esto implica debe hacer entrega de por lo menos cinco archivos `Kotlin`. Los nombres de esos cinco archivos son los siguientes: `Cancion.kt`, `Reproductor.kt`, `ListaReproduccion.kt`, `ArbolDeCanciones.kt` y `AdministradoDeMusica.kt`. Usted puede crear otros archivos en `Kotlin` si así lo considera necesario. No puede usar librerías de `Kotlin` que correspondan a contenedores o estructuras de datos contenedoras, salvo los arreglos, tuplas, y pares. Para la solución de este proyecto, debe implementar las estructuras de datos que utilice. Debe realizar un archivo `Makefile` que compile todos los archivos fuentes `Kotlin` de su programa. Debe entregar un archivo `README.md`, en formato Markdown, en el que describan cada uno de los archivos del proyecto y expliquen las estructuras de datos usadas y el diseño de su solución.

Para la implementación de TAD Reproductor se requiere el uso de una librería de audio para poder reproducir archivos en formato MP3. Para este proyecto se debe hacer uso de la librería, en `Java`, `PausablePlayer` [1]. Esta librería está basada en la librería, en `Java`, `JLayer` [2], agregando la función de pausar la reproducción de un archivo MP3. En la página del curso se publicarán las librerías `PausablePlayer` y `JLayer`, que son necesarias para su proyecto.

Su programa debe funcionar en la plataforma Linux. Si su aplicación no se ejecuta en Linux la nota del proyecto será *cero*. El código debe estar debidamente documentado y cada uno de los métodos de los TADs, y demás funciones y procedimientos, deben tener los siguientes elementos: descripción, parámetros, precondición y postcondición. Además debe hacer uso de la guía de estilo recomendada para `Kotlin`.

4. Condiciones de entrega

Debe entregar los códigos fuentes de su programa, y la declaración de autenticidad, en un archivo comprimido llamado `Proyecto2-X-Y.tar.xz` donde X y Y son los números de carné de los autores del proyecto. La entrega debe hacerse por medio de la plataforma Classroom, antes de las 8:50 PM del día viernes 05 de abril de 2024.

Guillermo Palma / gvpalma@usb.ve / Marzo 2024

[1] Disponible en `https://stackoverflow.com/questions/12057214/jlayer-pause-and-resume-song?noredirect=1&lq=1`
[2] Disponible en `https://web.archive.org/web/20210108055829/http://www.javazoom.net/javalayer/javalayer.html`