Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2692 - Laboratorio de Algoritmos y Estructuras II
Trimestre Enero-Marzo 2017

Proyecto 2: Lista de reproducción de música

Introducción

El objetivo de este proyecto es el desarrollo diferentes estructuras abstractas de datos para la representación de una lista de reproducción, con el objetivo de que al final lo incorpore al reproductor que se le brindará como base. Deberá leer desde un archivo la biblioteca de canciones disponibles e implementar los distintos métodos para su manipulación y consulta dentro del reproductor.

TAD Canción

TAD que representa a una canción a reproducir que se encontrará en el archivo `cancion.py`, contiene los atributos de instancia título, artista y archivo (de tipo wav). Donde el título y artita son strings no vacíos y el archivo es una instancia de QFile donde la dirección del archivo corresponde a uno de música de formato wav.

Atributos

- *titulo*. Atributo de instancia de tipo *string* que debe contener al menos un caracter.
- *artista*. Atributo de instancia de tipo *string* que debe contener al menos un caracter.
- *archivo*. Atributo de instancia que guarda un objeto de la clase *QFile* construido a partir de un *string* que representa la dirección de un archivo de audio.

Métodos

- $\_\_init\_\_(self, titulo, artista, archivo)$. Constructor de la clase que recibe un título y un artista, los cuales son *strings* no vacíos, además la dirección de un archivo en *string* que no puede ser `None`. Estos parámetros deben inicializar cada atributo de instancia correspondiente.
- $es\_igual(self, cancion)$. Compara otra instancia de la clase Cancion con *self* y verifica si son iguales por titulo y artista, retornando `True` en caso de serlos y `False` en caso contrario.
- $es\_menor\_titulo(self, cancion)$. Compara otra instancia de la clase *Cancion* con *self* y verifica si es menor por el titulo. Si el titulo es igual, compara por artista, retornando `True` en caso de que `self` sea menor y `False` en caso contrario.
- $es\_menor\_artista(self, cancion)$. Compara otra instancia de la clase *Cancion* con *self* y verifica si es menor por artista. Si el artista es el mismo, compara por titulo, retornando `True` en caso de que `self` sea menor y `False` en caso contrario.

  Nota: la comparación de titulo y artista es la que realiza *Python* entre *strings*.

Para obtener el valor almacenado en los atributos de instancia a continuación se muestra como se haría.

```
c = Cancion("Across The Universe","The Beatles","direccion/al/archivo.wav")

c.titulo #> "Across The Universe"
c.artsta #> "The Beatles"
c.archivo #> <Objeto QFile>
```

TAD Lista de reproducción

TAD que representa un lista de reproducción que se encontrará en el archivo `lista.py`. Esencialmente se debe implementar una lista circular doblemente enlazada [Cor09] y existirá un atributo de instancia que contiene una referencia a uno de los nodos, además debe mantenerse un atributo con el tamaño de la lista. Para poder implementar la lista se necesita un nodo que tenga los atributos *siguiente* y *anterior* que referencian al siguiente y anterior nodo. Por otro lado, el hecho de que sea circular indica que el atributo *siguiente* del último nodo apunta al primero y el atributo *anterior* del primer nodo apunta al último.

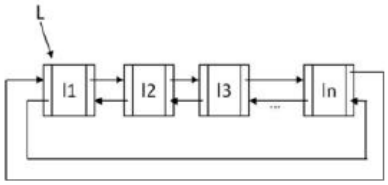

Figura 1. Lista circular doblemente enlazada

Atributos

- *proximo*. Atributo de instancia que guarda una referencia a la próxima canción a sonar
- *numero_nodos*. Atributo de instancia que indica la cantidad de canciones en la lista de reproducción.

Métodos

- $\_\_init\_\_(self)$. Constructor de la clase. Crea un lista vacía donde *proximo* sería `None` y *numero_nodos* sería `0`.
- $agregar(self, cancion)$. Inserta un nuevo objeto de la clase Cancion en la posición donde esté apuntado el atributo *proximo* de la lista. Es decir, el atributo *proximo* referenciaría a ese nuevo nodo.
- $agregar\_final(self, cancion)$. Inserta un nuevo objeto de la clase Cancion justo antes del nodo *proximo* de la lista.
- $ordenar\_titulo(self)$. Ordena las canciones de la lista de reproducción por título, usando el método *es_menor_titulo* de comparación de la clase Cancion con el algoritmo de ordenamiento *Mergesort*
- $ordenar\_artista(self)$. Ordena las canciones de la lista de reproducción por artista, usando el método *es_menor_artista* de comparación de la clase Cancion con el algoritmo de ordenamiento *Mergesort*.
- $eliminar(self, tituloCancion)$. Elimina la canción de la lista cuyo título sea igual al título de tipo *String* pasado como parámetro.

Cabe destacar que la lista no puede tener canciones repetidas, es decir, no debe existir ningún par de canciones que al momento de compararlas con el método *es_igual* retorne `True` .

TAD NodoLista

El TAD lista de reproducción usará una clase para representar los nodos, esta se encuentra implementada y suministrada junto al enunciado.

```
1     class NodoLista:
2         def __init__(self, e, s, a):
3             self.elemento = e
4             self.siguiente = s
5             self.anterior = a
```

TAD Reproductor

TAD ubicado en el archivo `reproductor.py`. A pesar de que no implementará ninguna interfaz gráfica, deberá instalar la librería PyQt5, en específico se usarán los módulos PyQt5.QtCore, PyQt5.Gui, PyQt5.QtMultimedia y PyQt5.QtWidgets, para correr el proyecto. Son esenciales para que se muestre la ventana donde estará la lista de reproducción y los distintos botones del reproductor.

Aunque no se modifique esta clase, deberá usar dos métodos de ella para manipular la lista de reproducción y que se visualice en la interfaz gráfica. Esos métodos son los siguientes.

- *sonarDespues*(*cancion*). Agrega una canción justo después de la canción actual. Este método utiliza el método *agregar* del TAD Lista de Reproducción.
- *sonarAntes*(*cancion*). Agrega una canción justo antes de la canción actual. Este método utiliza el método $agregar\_final$ del TAD Lista de Reproducción.

Módulo Cliente

El módulo cliente donde el qrchivo se llamará `cliente.py` será quien use las estructuras antes mencionadas para consolidar el reproductor de música, para ello se deberá implementar un menú por consola con varias opciones para consultar las canciones disponibles y manipular la lista de reproducción, además se tendrá que procesar un archivo de entrada donde estará la información de las canciones que estarán disponibles.

Menú

El objetivo es que se manipule la aplicación a través de la linea de comandos y por medio de una interfaz gráfica. La última parte ya está implementada, así que se requiere que desarrolle el menú de consola. La ejecución del programa será por medio del comando.

$$\$\ python3\ cliente.py\ <\ archivo\ >$$

En un principio, el menú mostrará las posibles opciones y luego solicitará que se introduzca un número correspondiente a la opción. En caso de no ser una opción válida, se ignorará. El menú sería similar al que se muestra a continuación.

```
1. Listar canciones
2. Agregar para sonar justo después de la canción actual
3. Agregar para sonar justo antes de la canción actual
4. Ordenar lista de reproducción por artista
5. Ordenar lista de reproducción por titulo
6. Eliminar canción por título
7. Mostrar opciones
8. Salir

Escoja una opción:
```

La lista de opciones sólo se mostrará al principio y si se solicita como opción. A continuación se detalla cada opción que debe implementar.

1. Se deben listar todas las canciones disponibles, en el orden que se encuentren.
2. Es necesario que el programa le pida al usuario los datos de una canción para luego crear una instancia `c` de la clase **Cancion** y agregarla en la lista de reproducción, y de último se llamaría al método de la clase Reproductor[1] *sonarDespues* que recibe una instancia de la clase **Cancion**.
3. Igual que en la opción anterior, a diferencia que se llama al método *sonarAntes*.
4. Ordena la lista de reproducción por artista.
5. Ordena la lista de reproducción por título.
6. Muestra las opciones disponibles.
7. Salir del reproductor.

Archivo de entrada

El archivo de entrada tendrá un formato donde la primero linea corresponde al número de canciones en el archivo, las próximas `n` linea contendrán un *string* compuesto por el nombre del artista, el titulo de la canción y la dirección del archivo de música y entre ellos existirá un caracter tabulador para separarlos. Puede darse el caso donde exista un salto de linea al final del *string*, el cual no pertenece a la dirección del archivo. A continuación se muestra el formato.

```
n
<artista>[TAB]<titulo>[TAB]<direccion de archivo>
<artista>[TAB]<titulo>[TAB]<direccion de archivo>
<artista>[TAB]<titulo>[TAB]<direccion de archivo>
...
<artista>[TAB]<titulo>[TAB]<direccion de archivo>
```

Deberá leer cada linea, instanciar la clase Cancion e ir almacenándolas en un *ArrayT*. Así se tendrán a disposición todas las canciones de lista de reproducción.

[1] La clase Reproductor no debe se modificada en ningún sentido, ella usa las clases que se piden en este enunciado

Condiciones de entrega

Su programa debe poderse ejecutarse en Python 3 en el sistema de operación `Linux`. Si un programa no se ejecuta, el equipo tiene cero como nota del proyecto. El trabajo es por equipos de laboratorio. Debe entregar los códigos fuentes de sus programas, en un archivo comprimido llamado Proyecto2-X-Y.tar.gz donde X y Y son los números de carné de los integrantes del grupo. La entrega se realizará por medio del aula virtual antes de la 11:59 pm del sabado 18 de Marzo de 2017. Cada grupo debe entregar firmada, en el laboratorio de la semana 10, la "Declaración de Autenticidad para Entregas", cuya plantilla se encuentra en la página del curso. Ambos integrantes del equipo deben trabajar en el proyecto. El no cumplimiento de algunos de los requerimientos podrá resultar en el rechazo de su entrega.

Referencias

[Cor09] T.H. Cormen. *Introduction to Algorithms*. MIT Press, 2009. ISBN: 9780262033848. URL: `https://books.google.co.ve/books?id=i-bUBQAAQBAJ`.