Isai Daniel Chacón Silva - 201912015

Nicolás Aparicio Claros - 201911357

Estructuras de datos y algoritmos

LabLists-S03-G01

Paso 3

- ¿Cuáles son los mecanismos de interacción (I/O: Input/Output) que tiene el **view.py** con el usuario?

Existe una variable de tipo string llamada *inputs* que se encarga de recibir la opción que el usuario selecciona del menú una vez se ejecuta el **view.py**.

En dado caso de que la variable *inputs* haya tomado el valor de "1", se inicializa el catálogo y el mecanismo de interacción de output vendrá dado por una serie de *prints* que informarán al usuario la cantidad de libros, autores, géneros y asociaciones de géneros a los datos que fueron cargados.

Si la variable *inputs* toma el valor "2", nuevamente se da un mecanismo de interacción de input al preguntar por los TOP que se desean buscar de los libros. Luego, se realiza un *output* que es un *print* que indica al usuario los mejores libros en el rango indicado o que no se encontraron libros en su defecto.

Por su parte, si la variable *inputs* toma el valor "3", se da un mecanismo de interacción mediante un input referente al nombre del autor que se desea buscar en la base de datos. Luego, se da un output al usuario en forma de *print* que describe el nombre del autor, el promedio registrado, el total de libros y el título de cada libro en conjunto con su ISBN. Si esta información no puede ser encontrada, esto también se le indica al usuario.

Finalmente, el último valor que puede adquirir la variable *inputs* es el de "4". De manera contraria, el programa cerrará su ejecución. Para este último caso, se pide nuevamente un *input* referente a la etiqueta a buscar de los libros. El output se realiza en forma de *print* informando la cantidad de libros en el catálogo para la etiqueta dada.

- ¿Cómo se almacenan los datos de **GoodReads** en el **model.py**?

La información de los datos de **GoodReads** se guarda en **model.py** a través de una función que ha sido designada como newCatalog(), esta crea un diccionario con 4 llaves y sus respectivos valores. La primera llave asignada es "books" y su valor corresponde a una lista con estructura de dato tipo "SINGLE_LINKED". Por otro lado, se tienen las claves de "authors", "tags" y "book_tags". Estas tres llaves tienen asignado a su valor una estructura de dato tipo "ARRAY_LIST". No obstante, difieren en cuanto al parámetro que indica la función de comparación a usar para los elementos de lista. Para el caso de "authors" se coloca *compareauthors* para este parámetro, en "tags" el valor asignado es *comparetagnames* y, finalmente, en el caso de la llave "book_tags" no se le asigna un valor en especial, por lo que la función de comparación en este escenario es *None.* Es importante mencionar que, para crear el diccionario con las claves y valores ya mencionados anteriormente, es necesario hacer uso del archivo *list.py*.

- ¿Cuáles son las funciones que comunican el **view.py** y el **model.py**?

La comunicación entre el view.py y el model.py se realizan mediante el controlador (controler.py), tal como indica la arquitectura MVC. A continuación, se presenta una tabla mediante las funciones que comunican estos dos módulos:

view.py	controller.py	model.py
initCatalog()	initCatalog()	newCatalog()
loadData(catalog)	loadData(catalog)	addBook(catalog, book) addTag(catalog, tag) addBookTag(catalog, booktag) sortBooks(catalog)
controller.getBestBooks(catalog, int(number))	model.getBestBooks(catalog, number)	getBestBooks(catalog, number)
controller.getBooksByAuthor(catalog, authorname)	model.getBooksByAuthor(catalog, authorname)	getBooksByAuthor(catalog, authorname)
controller.countBooksByTag(catalog, label)	model.countBooksByTag(catalog, tag)	countBooksByTag(catalog, tag)

Paso 4

- ¿Cómo se crea una lista?

Teniendo en cuenta que uno de los objetivos del laboratorio es la familiarización con el uso de TAD Lista, dentro del repositorio se encontraba un directorio denominado ADT, este contenía un archivo con el nombre de *list.py*. Dentro de *list.py* se encuentra definida newList(), una función que se encarga de crear una lista vacía. Para ello requiere de unos parámetros que tienen establecidos unos valores por defecto. Los parámetros y los valores asignados son: *datastructure='SINGLE_LINKED', cmpfunction=None, key=None, filename=None y delimiter=","*. Los argumentos de la función tienen un rol importante, ya que estos indican algunas características para la lista que será creada tal y como se muestra a continuación:

- *datastructure:* se encarga de definir el tipo de estructura de datos a utilizar a la hora de la creación de la lista.
- *cmpfunction:* es la función de comparación implementada para los elementos que componen a la lista.
- *key:* este parámetro permite hacer uso de un identificado que permitirá realizar la comparación de dos elementos de la lista a través de la función de comparación previamente definida.
- *filename:* en este caso, el valor asignado permite la creación de una lista teniendo en cuenta elementos encontrados en el archivo. Este archivo debe de ser *.csv*.
- *delimiter:* si hay un valor asignado al parámetro *filename* diferente de *None,* el valor asignado para *delimiter* permitirá realizar una separación de los campos a través de este carácter asignado. En este caso corresponde a una coma (,)

A través de la implementación de un *try* se busca la creación de una lista nueva con los parámetros establecidos y se llama a la función newList(*datastructure, cmpfunction, key, filename, delimeter*) del archivo *liststructure.py,* el cual se encuentra ubicado en DISClib/DataStructures. La función newList del módulo *liststructure.py* distingue entre el tipo de estructura de dato para la lista a crear. En caso de que sea "ARRAY_LIST" se procede a llamar al módulo *arraylist.py,* el cual tiene una función newList(*cmpfunction, key, filename, delim*). Esta función retorna un diccionario con 5 claves (*'elements', 'size', 'type', 'cmpfunction'* y *'key'*) con sus respectivos valores. Posteriormente se lee el archivo que tenga el nombre del parámetro *filename.* Lo anterior se realiza a través de la implementación de la función *csv.DictReader*. Una vez se encuentra el archivo y se lee, se realiza un ciclo que recorrerá línea por línea y, se hará uso de la función addLast(), la cual recibe el diccionario creado y la línea leída del archivo. Se añade el elemento en el valor de la llave '*elements'* del diccionario y se aumenta de uno en uno el valor de la clave 'size' del mismo. De esta manera, es posible ir añadiendo los elementos en la última posición de la lista. Al final se retorna el diccionario con la información actualizada.

Por otro lado, si el parámetro de *datastructure* da le función newList() es "SINGLE_LINKED", el módulo de *liststructure.py* llamará a *singlelinkedlist.py.* En este archivo se ha definido una función newList(*cmpfunction, key, filename, delim*). Al igual que la función newList() del módulo *arraylist.py,* el módulo *singlelinkedlist.py* retorna un diccionario con 5 claves (*'first', 'last', 'size', 'key'* y *'type'*) con sus respectivos valores. Posteriormente se lee el archivo que tenga el nombre del parámetro *filename.* Lo anterior se realiza a través de la implementación de la función *csv.DictReader*. Una vez se encuentra el archivo y se lee, se realiza un ciclo que recorrerá línea por línea y, se hará uso de la función addLast(), la cual recibe el diccionario creado y la línea leída del archivo. No obstante, la función addLast() del módulo *singlelinkedlist.py* difiere de la del módulo *arraylist.py,* ya que la del módulo *singlelinkedlist.py* crea un nuevo nodo a través de la función *node.newSingleNode()* con el elemento leído en el archivo. En la llave del diccionario creado en el nuevo nodo, se asigna en la clave *'next'* el primer elemento, es decir, el valor que está asignado a la llave *'first'* del diccionario de la función newList(). En caso de que el tamaño sea igual a 0 el valor de la llave *'last'* del diccionario corresponde al valor del primer elemento, es decir, el valor de *'first'*. De lo contrario, se aumenta el valor de *'size'* en uno. Finalmente se retorna el diccionario actualizado.

Es así como se pueden crear las listas dependiendo del tipo de dato que se quiera implementar.

- ¿Qué hace el parámetro cmpfunction = None en la función newList()?

Este parámetro dentro de la función newList() si es None le indica al programa que utilice como función de comparación aquella implementada por defecto para los elementos de la lista. Esta función recibe como parámetros dos id y los compara por su magnitud, devolviendo valores de 1 (id1 > id2), -1 (id < id2) o 0 (id1 = id2). También vale recalcar que para el caso en que cmpfunction es None, es necesario especificar el valor del parámetro key.

- ¿Qué hace la función addLast()?

Esta función recibe por parámetros la lista (lst) y el elemento (element) que desea ser añadido a esta en la última posición. La función realiza el try llamando a la función addLast() del módulo liststructure. Dentro de este módulo se verifica el tipo de la lista (ArrayList o SingleLinked) dado que para ambos casos se añade de una manera distinta. Para el caso del

ArrayList simplemente se llama a la función append y se aumenta el valor guardado en size. En el caso de la SingleLinked es necesario crear un nuevo nodo, verificar si es el primer nodo para inicializar la lista mediante la key asociada a “first”; o en caso contrario realizar las correspondientes asignaciones para que el penúltimo nodo apunte al nuevo último y este sea almacenado en el valor de la llave “last” al tiempo que se aumenta el valor asociado a size. En caso de que ocurra algún error, este método lanzará una excepción.

- ¿Qué hace la función getElement()?

Esta función recibe como parámetros la lista (lst) y la posición del elemento a buscar (pos). En este caso la lista no puede ser vacía, dado que se recorre hasta el elemento pos, el cual debería ser mayor a 0 y menor que el tamaño completo de la lista. La función retorna el elemento buscado sin eliminarlo. Así como en el caso anterior, se hace un llamado al módulo liststructure mediante la función getElement(). Para el ArrayList se busca en la llave asociada a elements que es una lista nativa de python, por lo que esta simplemente se indexa en pos-1. En el caso de la SingleLinked se avanza en los nodos mediante un ciclo while y la instrucción node[“next”] hasta que el contador iguale el valor de pos. En caso de que ocurra algún error, este método lanzará una excepción.

- ¿Qué hace la función subList()?

La función subList() tiene como objetivo generar una sublista de la lista grande que contiene toda la información. SubList () tiene 3 argumentos. El primero de ellos es *‘lst’*, este corresponde a la lista que se quiere recorrer, usar. De esta manera, idealmente se espera que esta lista contenga información, es decir, que no esté vacía. El segundo parámetro es *‘pos’*, este se refiere al número de la posición desde la cual se quiere empezar a crear la sublista. Finalmente, se encuentra *‘numelem’* que hace referencia al número de elementos que se quieren copiar en la sublista a realizar. Para cumplir con el objetivo de subList() se llama a la función subList() del módulo *liststructure.py.*

Paso 5

- ¿Observó algún cambio en el comportamiento del programa al cambiar la implementación del parámetro “ARRAY_LIST” a “SINGLE_LINKED”?

Inicialmente se tuvo en cuenta el parámetro de tiempo a la hora de cargar los datos dependiendo de la estructura que se quería implementar. Cuando se seleccionaba “ARRAY_LIST” se obtuvo un tiempo de 13 minutos con 6 segundos y 96 milisegundos mientras que para un tipo de dato “SINGLE_LINKED” el tiempo fue de 10 minutos con 13 segundos y 21 milisegundos. Es importante mencionar que la carga de datos correspondió a los archivos que **no** eran la versión -small. Adicionalmente, con las versiones -small de los archivos, decidimos determinar si la cantidad de archivos cargados en ambos tipos de datos (“ARRAY_LIST” y “SINGLE_LINKED”) correspondían al mismo valor. De esta manera, para ambos casos, se reportaron: un total de 149 libros, 156 autores, 34252 géneros y 999 asociaciones de géneros a los libros cargados. De acuerdo con lo anterior, es posible mencionar que, sin importar la estructura del tipo de dato, toda la información cargada será la misma en ambos casos. Adicionalmente, se puede concluir que las observaciones realizadas hacen que el tipo de dato “SINGLE_LINKED” sea óptimo a la hora de cargar los datos de este laboratorio.

Finalmente, de manera teórica esto también se puede justificar, ya que, en el caso de una lista enlazada, la memoria se asigna a medida que se añaden los datos a la lista, en otras palabras, durante el tiempo de ejecución del algoritmo. Por otro lado, las listas tipo array consumen ubicaciones de memoria contiguas asignadas en tiempo de compilación, es decir, durante la declaración del array [1].

Referencias:

[1] Studytonight. n.d. Difference between Array and Linked List. [online] Available at: <https://www.studytonight.com/data-structures/linked-list-vs-array#:~:text=Both%20Linked%20List%20and%20Array,it%2C%20which%20means%20at%20runtime.> [Accessed 1 September 2021].