OBSERVACIONES DE LA PRACTICA

Ana Sofía Castellanos 202114167

Martín Santiago Galván Castro 201911013

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

|  |  |  |
| --- | --- | --- |
| Posición | Input | Output |
| Menú  principal | Pide al usuario el npumero de la opción que desea ejecutar. | Muestra al usuario las opciones de las funciones que puede elegir.  Dirige al usuario a la opción elegida. |
| Cargar la información | No solicita nada al usuario | Muestra un mensaje con la siguiente información:  “””  Cargando información de los archivos ....  Libros cargados: \_\_\_  Autores cargados:\_\_\_  Géneros cargados: \_\_\_  Asociación de Géneros a Libros cargados: \_\_\_  “”” |
| Top x | Solicita la cantidad de libros que se desea consultar en la lista de Ranking | Muestra un mensaje con los títulos de la cantidad de libros solicitados por el usuario que se ubican en el top:  “””  Estos son los mejores libros:  Titulo: \_\_\_\_\_\_  Titulo: \_\_\_\_\_\_  …  “”” |
| Libros de un autor | Solicita el nombre del autor a consultar | Muestra un mensaje con el nombre del autor, el promedio, la cantidad de libros y los libros encontrados del autor:  “””  Autor encontrado:\_\_\_\_\_  Promedio:\_\_\_\_\_  Total de libros:\_\_\_\_\_  Titulo:\_\_\_\_\_  Titulo:\_\_\_\_\_  …  “”” |
| Libros por género | Solicita la etiqueta (género) a consultar | Muestra un mensaje con la cantidad de libros de esa categoría:  “””  Se encontraron \_\_\_ libros”  “”” |

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

Por medio de la función newCatalog(), se crea un catalogo de libros en donde se almacena la información. Este catologo es un diccionario con las llaves: ‘books’, authors’´, ‘tags’, y ‘book\_tags’. A cada llave se le asigna una lista TAD vacía. En el documento original , todas las listas son declaradas como arreglos menos libros, la cual no tiene declarado un tipo de estructura de datos y por defecto los guarda como una lista encadenada.

Los libros son almacenados en la lista de la llave ‘books’ del catálogo, cada nodo cuenta con toda la información del libro.

Los autores son almacenados en la lista de la llave ‘authors’ del catálogo, esta lista a su vez guarda referencias de los libros de ese autor.

Los tags son almacenados en la lista de la llave ‘tags’ del catálogo, cada uno con la forma de un diccionario de la forma {‘name’:\_\_\_ ,‘tag\_id’:\_\_\_}.

Los book-tags son almacenados en la lista de la llave ‘book\_tags’ del catálogo, cada uno con la forma de un diccionario de la forma {‘tag\_id’:\_\_\_ ,‘book\_id’:\_\_\_}.

1. ¿Cuáles son las funciones que comunican el view.pyy el **model.py**?

Las funciones que comunican view.py con model.py son las funciones que se encuentran en controller.py. Más especificamente, las funciones de view.py llaman las funciones que se encuentran en controller.py, las cuales a su vez, llaman las funciones del modelo.

En view.py, las funciones initCatalog() y loadData(catalog) usan funciones que se encuentran en el controlador, controller.py, estas funciones del controlador usan a su vez funciones del modelo. Adicionalmente, en view.py, se usan funciones del controlador que no estan contenidas en otras funciones independientes, estas son getBestBooks(…), getBooksByAuthor(…), countBooksByTag(…); estas funciones comunican el controlador con el modelo.

1. ¿Cómo se crea una lista?

En el archivo list.py, se crean las listas como una lista vacía, especificamente como una lista simplemente encadenada por default. Esto se hace con la función newList(), la cual tiene como parámetros ‘datastructure’ la cual se especifica en este caso como ‘SINGLE\_LINKED’, ‘cmpfunction’ = None, key = None, filename = None, delimiter=’,’.

Esta función hace uso de la función newlist(…) de un archivo llamado liststructure.py en la carpeta DataStructures. Esta función a partir del parámetro ‘datastructure’ de la función newlist() del archivo list.py llama a otra función de otro archivo, si ‘datastructure’ == ARRAY\_LIST llama a la función newlist(…) del archivo arraylist.py, de lo contrario llama a la función newlist del archivo singlelinkedlist.py.

Como en este caso ‘datastructure’ == ’SINGLE\_LINKED’, se llama a la función newlist() del archivo singlelinkedlist.py, el cual crea una nueva lista vacía como un diccionario el cual contiene como llaves: 'first': None, 'last':None, 'size': 0, 'key': key y 'type': 'SINGLE\_LINKED'.

En el caso en el que ‘datastructure’ == ‘ARRAY\_LIST”, se llama a la función newlist() del archivo arraylist.py, el cual crea una nueva lista vacía como un diccionario el cual contiene como llaves: 'elements': [], 'size': 0, 'type': 'ARRAY\_LIST', 'cmpfunction': cmpfunction y 'key': key.

1. ¿Qué hace el parámetro **cmpfunction=None** en la función **newList()**?

Según la documentación, cmpfunction se refiere a la función de comparación para los elementos de la lista. Si no se provee función de comparación se utiliza la función por defecto.

En este caso, dado que se especifica None para la función de comparación, queda como función de comparación una por defecto: la función defaultfunction(id1,id2), el cual compara y devuelve: 1 en el caso en el que id1 > id2, -1 en el caso en el que id1 < id2, y 0 si son iguales.

1. ¿Qué hace la función **addLast()**?

Según la documentación, la función addLast(…) adiciona un elemento en la última posición de la lista y actualiza el apuntador a la ultima posición (en caso de ser una lista encadenada). Se incrementa el tamaño de la lista en 1.

La función addLast(…) del archivo list.py invoca a la función addLast(…) del archivo liststructure.py, la cual llama a otra función de otro archivo según el tipo de estructura de dato que tenga la lista, así si es un ARRAY\_LIST llama a la función addLast(…) del archivo arraylist.py y si es SINGLE\_LINKED a la función addLast(…) del archivo singlelinkedlist.py.

La función addLast(…) del archivo arraylist.py recibe como parámetros la lista y el elemento a añadir, esta lista a su vez tiene la estructura de un diccionario con diversas llaves entre ellas ‘elements’:[...] y ‘size’: (con un número entero con la cantidad e elementos de la lista). La función addLast añade el elemento a la lista de la llave ‘elements’(lista[‘elements’].append(element)) e incrementa el tamaño de la lista en uno.

La función addLast(…) del archivo siglelinkedlist recibe como parámetros la lista y el elemento a añadir. Con el elemento crea en primer lugar un nodo con la función newSingleNode(element) que llama desde el archivo listnode.py, este nodo es un diccionario que contiene como llaves a ‘info’:element, ‘next’: None. En segundo lugar, la función addLast(…) verifica si la lista es vacía, de ser así asigna el primer elemento de la lista como el nodo creado, de lo contrario actualiza el apuntador del último elemento (antes de agregar el nuevo), es decir el nodo ‘next’como el nodo creado. Tras ello, se actualiza el último elemento de la lista como el nodo creado y se incrementa el tamaño de la lista en uno.

1. ¿Qué hace la función **getElement()**?

Según la documentación, la función getElement(…) retorna el elemento que se encuentra en la posición dada por parámetro de la lista dada por parámetro. Para ello recorre la lista hasta llegar a dicho elemento, cuyo índice de posición debe ser mayor a cero y menor o igual al tamaño de la lista. Retorna el elemento sin eliminarlo.

La función getElement(…) del archivo list.py invoca a la función getElement(…) del archivo liststructure.py, la cual llama a otra función de otro archivo según el tipo de estructura de dato que tenga la lista, así si es un ARRAY\_LIST llama a la función getelement(…) del archivo arraylist.py y si es SINGLE\_LINKED a la función getelement(…) del archivo singlelinkedlist.py.

La función getElement(…) del archivo arraylist.py recibe como parámetros la lista y la posición a consultar, al ser un arreglo para hallar al elemento solicitado la función saca la llave ‘elements’ de la lista, la cual contiene una lista con los elementos de la lista y extrae el elemento de la posición ingresada por parámetro.

La función getElement(…) del archivo singlelinkedlist.py recibe como parámetros la lista y la posición a consultar, al ser una lista encadenada para hallar el elemento solicitado la función crea un ciclo para buscar el elemento nodo a nodo hasta encontrar la posición deseada, para ello inicia la busqueda en el primer nodo y a partir de allí recorre la lista hasta llegar al elemento deseado y devuelve la llave ‘info’ del nodo solicitado.

1. ¿Qué hace la función **subList()**?

Según la documentación, la función subList(…) retorna una lista más pequeña que la lista original que contiene los elementos desde una posición dada por parámetro, con un número de elementos dados por parámetro.

La función sublist(…) del archivo list.py invoca a la función sublist(…) del archivo liststructure.py, la cual llama a otra función de otro archivo según el tipo de estructura de dato que tenga la lista, así si es un ARRAY\_LIST llama a la función sublist(…) del archivo arraylist.py y si es SINGLE\_LINKED a la función sublist(…) del archivo singlelinkedlist.py.

La función subList(…) del archivo arraylist.py recibe como parámetros la lista, la posición desde que se hará la nueva sublista y la cantidad de elementos de la nueva sublista. La función crea una sublista vacía en forma de diccionario que contiene las llaves ‘elements’:[], ‘size:0, 'type': 'ARRAY\_LIST', 'key': lst['key'], 'cmpfunction': lst['cmpfunction'], donde key y cmpfunction tienen como valores los mismos de la lista original. Tras ello la función realiza un ciclo para agregar a la llave ‘elements’ de la sublista los elementos de la lista original a partir de la posición ingresada por parámetro hasta que se complete la cantidad de elementos ingresada por parámetro. Finalmente, devuelve la sublista con los elementos seleccionados de la lista orginal.

La función subList(…) del archivo siglelinkedlist recibe como parámetros la lista, la posición desde que se hará la nueva sublista y la cantidad de elementos de la nueva sublista. La función crea una sublista vacía en forma de diccionario que contiene las llaves 'first': None, 'last': None, 'size': 0, 'type': 'SINGLE\_LINKED', 'key': lst['key'], 'cmpfunction': lst['cmpfunction'], donde key y cmpfunction tienen como valores los mismos de la lista original. Tras ello, la función realiza un ciclo para encontrar cada elemento de la lista original a partir de la posición ingresada por parámetro a través de la función getElement(…) y añadir ese elemento al final de la nueva sublista creada mediante la función addLast(…). Finalmente, devuelve la sublista con los elementos que se añadieron de la lista original.

1. ¿Observó algún cambio en el comportamiento del programa al cambiar la implementación del parámetro **“ARRAY\_LIST”** a **“SINGLE\_LINKED”**?

Para analizar los cambios en el comportamiento del programa se analizará el tiempo que le toma al computador dar una respuesta al usuario tanto para la lista encadenada como para el arreglo.

Conjunto de datos pequeño: Conjunto de datos mediano:

|  |  |
| --- | --- |
| Tiempo de ejecución del programa- Books medium (ms) | |
| Array list | Single linked list |
| Cargar información | |
| 558.51 | 602.38 |
| Consultar los Top x libros | |
| 1041.90 | 809.17 |
| Consultar los libros de un autor | |
| 5296.88 | 7162.32 |

|  |  |
| --- | --- |
| Tiempo de ejecución del programa- Books small (ms) | |
| Array list | Single linked list |
| Cargar información | |
| 98.73 | 115.71 |
| Consultar los Top x libros | |
| 1618.64 | 1016.08 |
| Consultar los libros de un autor | |
| 1574.06 | 6574.29 |

A partir de los datos recolectados del tiempo de ejecución del programa se puede evidenciar que cargar los documentos toma un tiempo similar, aunque para la lista encadenada es ligeramente mayor. Por otro lado, el Arraylist es un poco más demorado para consultar los Top x de libros en comparación del Single linked list. De igual manera, la lista sencillamente encadenada le toma un tiempo mucho mayor devolver los libros de un autor en comparación del arreglo.