OBSERVACIONES DEL LA PRACTICA

Estudiante 1: Daniel Felipe Vargas, Cod. 202123899

Estudiante 2: Santiago Martínez, Cod. 202124032

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

La interacción con el usuario comienza con la ejecución de la función printMenu, mediante la cual se le da a conocer al usuario cuales son las funcionalidades que el programa tiene disponibles y cómo ejecutarlas. Dependiendo del input del usuario (1, 2, 3, 4, etc) view ejecutará un función distinta para cumplir con lo requerido por el usuario. Cabe mencionar que en cada funcionalidad, view se comunica con controller, que a su vez se comunica con model para cumplir con lo requerido. Por último, es importante mencionar que view también cuenta con ciertas funciones para moderar el output que se le muestra al usuario. Por ejemplo, las funciones printAuthorData y printBestBooks toman un grupo determinado de datos y los organizan de manera que, al imprimirlos, se vean de manera presentable y comprensible para el usuario.

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

En primera instancia, se crea un diccionario madre llamado catalog (que a su vez está contenido, en view y controller, en el diccionario control bajo la llave ‘model’). Este diccionario contiene cuatro llaves: ‘books’, ‘authors’, ‘tags’ y ‘book\_tags’. En cada llave se guarda una lista (TAD definido en ela carpeta DISClib/ADT) diferente que contiene información respectiva a lo indicado por cada llave. Es decir, ‘books’, ‘tags’ y ‘book\_tags’ contienen la información encontrada en su respectivo archivo csv. ‘authors’ es especial en el sentido de que se encuentra directamente relacionada con ningún archivo csv, sino que por el contrario se construye a partir de los datos encontrados en ‘books’. Esta última lista contiene múltiples diccionarios cada uno con el nombre de un autor, una lista de los libros publicados por dicho autor y su rating promedio. Por último cabe destacar que cada una de las listas en catalog usa una estructura de datos distinta: ‘authors’ y ‘tags’ usan ‘SINGLE\_LINKED’; ‘books’ y ‘book\_tags’ usan ‘ARRAY\_LIST’.

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

Todas las funciones encontradas en controller.py comunican el view.py con el model.py. Cabe señalar que algunas funciones como loadData no comunican ambos directamente, sino indirectamente. Es decir, son llamadas por view y en lugar de llamar directamente a model, llaman a otras funciones dentro de controller que a su vez se comunican con model.

1. ¿Cómo se crea una lista?

Para crear una nueva lista se utiliza la función **newList ()** en la localización DISClib/ADT/list.py, la cual toma como parámetro el tipo de estructura que el usuario desea que la lista utilice, la función de comparación necesaria para poder comparar los elementos dentro de la lista y el identificador de la función de comparación por defecto en caso de que esta sea la que seleccionó el usuario. Opcionalmente, el usuario también puede pasar el nombre de un archivo de tipo CSV y el delimitador usado en este archivo para crear una lista a partir de los archivos presentes en ese archivo.

Una vez el usuario le haya pasado estos datos al programa, la función **newList ()** le pasará estos datos a una segunda función **newList ()** ubicada en la localización DISClib/DataStructures/liststructure.py. Esta función a pesar de tener el mismo nombre cumple un propositivo muy distinto, ya que esta función tiene la única tarea de pasarle la información que nos entregó el usuario a la implementación de la estructurara correcta de acurdo a lo que el usuario haya pasado previamente por parámetro. Habiendo llegado a este paso, el programa se encargara de construir la lista de acuerdo a la implementación definida previamente por el programador con los parámetros pasados por el usuario. Una vez la lista ha sido creada, la función **newList ()** en liststructure.py le retorna la lista a la función **newList ()** en list.py, la cual le retorna la lista al usuario.

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

Este parámetro es necesario para que el usuario pueda ingresar la función de comparación que desea utilizar en su lista. Es posible observar que el valor de esta variable es de None por defecto, lo cual significa que por defecto la función de comparación a usar e la función por defecto

La función denominada “por defecto” se trata de la función defaultfunction, la cual devuelve un valor de 1, 0 ó -1 a la hora de comparar dos elementos, dependiendo de si el segundo elemento es mayor, igual o menor que el primero respectivamente.

1. ¿Qué hace la funció **a****ddLast()**?

La función a**ddLast()** se encarga de añadir a un elemento al final de una lista que el usuario pasa por parámetro a la función. El elemento que el usuario quiere añadir también se debe pasar por parámetro a la función **a****ddLast().**

Para añadir a un elemento, se pasa la información dada por parámetro a una función del mismo nombre en la localización DISClib/DataStructures/liststructure.py. Tal y como se mencionó previamente, las funciones en este archivo tienen la única tarea de pasarle los datos dados por el usuario a la implementación del tipo de lista del cual se pasó por parámetro, la cual se encargará de realizar la operación solicitada.

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

La función getElement() se encarga de retornar el valor de un elemento en la lista que este en la posición que el usuario ingrese por parámetro. Lógicamente, la posición que el usuario ingrese solo puede estar entre 0 y el tamaño de la lista, ya que de lo contrario esta no seria una posición valida dentro de la lista. Adicionalmente, el usuario debe pasar la lista donde desea buscar el elemento por parámetro.

Para buscar un elemento en la lista se pasa la información dada por parámetro a una función del mismo nombre **getElement()** en la localización DISClib/DataStructures/liststructure.py. Tal y como se mencionó previamente, las funciones en este archivo tienen la única tarea de pasarle los datos dados por el usuario a la implementación del tipo de lista del cual se pasó por parámetro, la cual se encargará de realizar la operación solicitada.

En caso de que la posición dada por el usuario sea valida, la función **getElement() r**etornara el valor que se encontraba en esa posición en la lista, de lo contrario mostrara un error.

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

La función **subList()** retorna una sub lista de una lista pasada por paramero. Esta sub lista consiste de los primeros n1 elementos que se encuentren después de la posición n2. La función llama a este n1 “numelem” y a n2 le denomina “pos”. Esta sub lista se retorna al usario si las pociciones dadas son validas.

Para extraer esta sub lista de la lista orignial se pasa la información pasada por parámetro a una función del mismo nombre subList() en la localización DISClib/DataStructures/liststructure.py. Tal y como se mencionó previamente, las funciones en este archivo tienen la única tarea de pasarle los datos dados por el usuario a la implementación del tipo de lista del cual se pasó por parámetro, la cual se encargará de realizar la operación solicitada.

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

Al cambiar este parámetro, no se genera ningún cambio que sea visible por el usuario en el funcionamiento del programa. Esto se debe a que lo único que cambia es como funciona la lista y como esta esta estructurada. Ya que la lista de tipo **“ARRAY\_LIST”** y **“SINGLE\_LINKED”** tienen los mismos métodos disponibles, no cambia la forma como el usuario utiliza la lista ni las operaciones que el usuario puede hacer sobre la lista. En síntesis, solamente se cambia la estructura de la lista pero al fin y al cabo la forma como el usuario utiliza la lista es la misma, por lo que no hay ningún cambio perceptible en el programa.