Estructuras de Datos y Algoritmos

Jose Luis Tavera - 201821999
Juan Diego Yepes - 202022391

Laboratorio II

Índice:

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

Analizando el código del archivo view.py, encontramos el siguiente fragmento de código que dictamina la interacción (el input y output) con el usuario actuando de forma de consola. La primera función define el menú que provee las opciones al usuario.

```
def printMenu():
    print("Bienvenido")
    print("1- Cargar informaci n en el catalogo")
    print("2- Consultar los Top x libros por promedio")
    print("3- Consultar los libros de un autor")
    print("4- Libros por g nero")
    print("0- Salir")
```

La función que se ejecuta al final del código, empieza generando un loop con la línea uno de "while True:" invocando la función de printMenu() definida previamente la cual imprime las opciones del view.py:

```
while True:
    printMenu()
    inputs = input('Seleccione una opci n para continuar\n')
```

Posteriormente, el usuario indica la función que quiere ejecutar, digitando el número correspondiente de "printMenu()", definida en la variable de inputs bajo la función de nombre análogo . En cada de las funciones subsecuentes se invocan funciones definidas en el controller.py para llevar a cabo las siguientes tareas:

Si el usuario digita la opción 1, la función que se ejecutará es la siguiente:

```
if int(inputs[0]) == 1:
        print("Cargando informacion de los archivos ....")
        catalog = initCatalog()
        loadData(catalog)
        print('Libros cargados: ' + str(lt.size(catalog['books'])))
        print('Autores cargados: ' + str(lt.size(catalog['authors'])))
        print('Generos cargados: ' + str(lt.size(catalog['tags'])))
        print('Asociacion de Generos a Libros cargados: ' +
              str(lt.size(catalog['book_tags'])))
```

Esta función indica la cantidad de libros, autores y géneros cargados en los archivos csv.

Por otro lado, cuando el usuario digita el número 2, se ejecuta la función descrita a continuación:

```
elif int(inputs[0]) == 2:
        number = input("Buscando los TOP ?: ")
        books = controller.getBestBooks(catalog, int(number))
        printBestBooks(books)
```

La función anterior consulta el top los mejores libros, que son los libros con mejor rating según el archivo csv, e imprime el número de libros indicado por el usuario.

Por otra parte, si el usuario digitara el número 3, la función que se ejecuta sería la siguiente:

```
elif int(inputs[0]) == 3:
        authorname = input("Nombre del autor a buscar: ")
        author = controller.getBooksByAuthor(catalog, authorname)
        printAuthorData(author)
```

Esta función busca los libros escritos por el autor delimitado por el usuario, y los imprime.

Si el usuario digita el número 4, se ejecutará la siguiente función:

```
elif int(inputs[0]) == 4:
        label = input("Etiqueta a buscar: ")
        book_count = controller.countBooksByTag(catalog, label)
        print('Se encontraron: ', book_count, ' Libros')
```

Invocando una función que retorna el número total de libros.

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

En principio, el model.py crea un catálogo que almacena la información contenida en el archivo .csv, creando un diccionario de TAD listas.

```
def newCatalog():
    """
    Inicializa el catalogo de libros. Crea una lista vacia para guardar
    todos los libros, adicionalmente, crea una lista vacia para los autores,
    una lista vacia para los generos y una lista vacia para la asociacion
    generos y libros. Retorna el catalogo inicializado.
    """
    catalog = {'books': None,
               'authors': None,
               'tags': None,
               'book_tags': None}

    catalog['books'] = lt.newList()
    catalog['authors'] = lt.newList('ARRAY_LIST',
                                    cmpfunction=compareauthors)
```

```
    catalog['tags'] = lt.newList('ARRAY_LIST',
                                 cmpfunction=comparetagnames)
    catalog['book_tags'] = lt.newList('ARRAY_LIST')

    return catalog
```

En este caso, el parámetro de la función lt.newList() es 'ARRAY_LIST', lo que configura la estructura de datos de arreglo. Lo anterior, es hecho con la función de newList() de la librería de DISCLib.

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

En el modelo de trabajo MVC, existe un archivo controller.py que comunica al view.py con el model.py. Esto se logra referenciando las funciones necesarias para que el código funcione. Un ejemplo es la siguiente operación, que busca los mejores libros de acuerdo al rating.

En primer lugar, en view.py el usuario indica que ésta es la función que desea ejecutar, esto lo explicamos anteriormente.

```
elif int(inputs[0]) == 2:
        number = input("Buscando los TOP ?: ")
        books = controller.getBestBooks(catalog, int(number))
        printBestBooks(books)
```

En segundo lugar, y como se evidencia en el código, se referencia la función getBestBooks() que se encuentra en controller.py al guardarla en la variable books.

```
def getBestBooks(catalog, number):
    """
    Retorna los mejores libros
    """
    bestbooks = model.getBestBooks(catalog, number)
    return bestbooks
```

En tercer lugar, se referencia la función la función getBestBooks() que se encuentra en el model.py y se guarda en la variable bestbooks.

```
def getBestBooks(catalog, number):
    """
    Retorna los mejores libros
    """
    books = catalog['books']
    bestbooks = lt.newList()
    for cont in range(1, number+1):
        book = lt.getElement(books, cont)
        lt.addLast(bestbooks, book)
    return bestbooks
```

Finalmente, esta función ya llega al catálogo donde encontramos la información requerida, y como podemos ver, es la que realiza todas las operaciones de agregar a la lista los libros iterando en el archivo csv.

Esto sucede de forma similar con varias de las otras funciones que encontramos en el app.

4 ¿Cómo se crea una lista?

Anteriormente, se analizaron las funciones de los módulos principales de nuestro sistema MVC (model.py, view.py y controller.py). No obstante, es evidente que en la función que genera el catálogo en el modelo "newCatalog()" se hace uso de la librería DISCLib, en esta encontramos la función de newList() que es la encargada de crear las listas:

```
def newList(datastructure='SINGLE_LINKED',
            cmpfunction=None,
            key=None,
            filename=None,
```

```
            delimiter=","):
    try:
        lst = lt.newList(datastructure, cmpfunction, key, filename, delimiter)
        return lst
    except Exception as exp:
        error.reraise(exp, 'TADList->newList: ')
```

Tal y cómo se puede apreciar, la función de newlist guarda los datos inicialmente como un TAD. Los cuales, se pueden específicar bajo los cinco argumentos de la función. Siendo el más importante el "datastructure" que señala el tipo de estructura de datos a utilizar para implementar la lista. Los tipos posibles pueden ser: ARRAY_LIST y SINGLE_LINKED.

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

Uno de los argumentos de la función newList() es cmpfunction. Esta es una función de comparación de los elementos de la lista - tal y como se describe posteriormente en los comentarios del código- es decir, su utilidad es especificar alguna función de comparación. En el caso de que no que se detalle cúal, ésta utilizará la función por defecto.

6 ¿Qué hace la función addLast()?

La función addLast() se muestra a continuación:

```
def addLast(lst, element):
    try:
        lt.addLast(lst, element)
    except Exception as exp:
        error.reraise(exp, 'TADList->addLast: ')
```

Esta función debe hacer varios procesos. En esencia lo que hace es agregar un dato al final del TAD lista. Para que esto sea posible, primero se busca el último elemento en la lista, luego, se adiciona este elemento, se actualiza el apuntador a la última posición y finalmente se incrementa el tamaño de la lista en 1.

7 ¿Qué hace la función getElement()?

La función getElement() se muestra a continuación:

```
def getElement(lst, pos):
    try:
        return lt.getElement(lst, pos)
    except Exception as exp:
        error.reraise(exp, 'List->getElement: ')
```

Esta función busca un elemento que se encuentre en la posición definida por el parámetro pos. Para lograrlo, recorre la lista hasta llegar a la posición indicada, y luego retorna ese elemento sin eliminarlo.

8 ¿Qué hace la función subList()?

La función sublist tiene tres parámetros: i) lst, ii) pos, iii) numelem.

```
def subList(lst, pos, numelem):
    try:
        return lt.subList(lst, pos, numelem)
    except Exception as exp:
        error.reraise(exp, 'List->subList: ')
```

Su función es retornar una nueva lista a partir de una lista indicada como lst, desde una posición específica (pos) y de una longitud predefinida (lst)

9 ¿Observó algún cambio en el comportamiento del programa al cambiar la implementación del parámetro "ARRAY_LIST" a "SINGLE_LINKED"?

```
def newCatalog():
    catalog = {'books': None,
               'authors': None,
               'tags': None,
               'book_tags': None}

    catalog['books'] = lt.newList()
    catalog['authors'] = lt.newList('SINGLE_LINKED',
                                    cmpfunction=compareauthors)
    catalog['tags'] = lt.newList('SINGLE_LINKED',
                                 cmpfunction=comparetagnames)
    catalog['book_tags'] = lt.newList('SINGLE_LINKED')

    return catalog
```

Gracias a la implementación del modelo MVC, en esencia no cambia nada en el comportamiento del programa, es decir, los resultados siguen siendo los mismos, ya que al cambiar la estructura de datos, solo se está modificando la manera en que se organizan estos datos, y nada a nivel de operaciones o datos. Sin embargo, puede que el orden de crecimiento al cambiar de estructura de datos sea diferente, ya que realiza diferentes operaciones para llegar a un mismo resultado.