OBSERVACIONES DE LA PRACTICA

Sebastian Martinez Arias Cod 202312210
Camilo Molina Plata 2 Cod 202221119
Samuel Rozen Cod 202123592

Preguntas de análisis

1) ¿Qué modificaciones debería hacer en la librería DISCLib para crear un nuevo algoritmo de ordenamiento?, especifique archivos, rutas de y encabezados de las posibles funciones a implementar.

Para crear un nuevo algoritmo de ordenamiento en la librería DISCLib, se deben seguir los siguientes pasos:

1. Definir el Algoritmo de Ordenamiento:
Primero, se debe tener una idea clara del algoritmo de ordenamiento que deseas implementar. Por ejemplo, si deseas implementar el algoritmo de ordenamiento por inserción, debes conocer su lógica y cómo funciona.

2. Modificar la Librería DISCLib:

a. Archivo de Algoritmo de Ordenamiento:
- Ubicación: `DISCLib/Algorithms/Sorting/`
- Se debe crear un nuevo archivo para tu algoritmo de ordenamiento, por ejemplo, `insertionsort.py` si se quiere implementar el ordenamiento por inserción.
- Dentro de este archivo, se implementa el algoritmo de ordenamiento.

b. Archivo `__init__.py`:
- Ubicación: `DISCLib/Algorithms/Sorting/`
- Se debe importar el nuevo algoritmo de ordenamiento en este archivo para que esté disponible para otros módulos. Por ejemplo, añadir `from . import insertionsort`.

3. Usar el Nuevo Algoritmo de Ordenamiento en la Aplicación:
- En la función `sortBooks`, en lugar de usar `qs.sort`, se puede usar el nuevo algoritmo. Por ejemplo, si se implementó el ordenamiento por inserción, se podria tener algo como `insertionsort.sort(books, compareISBN)`.

4. Criterio de Ordenamiento:
- La función `compareISBN` compara dos libros por su ISBN. Si se desea cambiar el criterio de ordenamiento, simplemente se modifica esta función. Por ejemplo, si se desea ordenar por título, se compara `book1["title"]` y `book2["title"]`.

En resumen, para añadir un nuevo algoritmo de ordenamiento a DISCLib:

- Se añade el algoritmo en un nuevo archivo dentro de `DISCLib/Algorithms/Sorting/`.
- Importa este algoritmo en el archivo `__init__.py` de la misma carpeta.
- Usar este algoritmo en la aplicación donde se necesite ordenar datos.

2) ¿Cómo se relaciona la opción 10 del menú en el **view.py** con las funciones principales (opciones 7, 8 y 9) dentro del **model.py**?

La relacion que tiene la opción 10 del menú con las funciones 7,8 y 9 es que, tras seleccionar esta opción, el usuario puede escoger como quiere que se solucione el problema presente del código, ya sea usando una función iterativa o una función recursiva, de tal modo se puede a su vez observar el rendimiento de ambas funciones en el código

3) ¿Por qué se usa la máscara en las funciones propuestas?, ¿siempre es necesaria?, argumente su respuesta.

El uso de una función según vemos en el proyecto hace varias cosas más fáciles, como por ejemplo el hacer los datos mucho más fáciles de organizar y a su vez ayudar a entender estos mismos, del mismo modo hay casos en los que hace que sea más fácil llamar unas funciones, por ejemplo cuando definimos que si es verdadero o falso el valor de recursividad no tenemos que dirigir a una u otra función en específico, sino que ambas entran por la máscara y en esta decide a que función redireccionara lo que se espera de resultado.

4) ¿Cuál es la causa del error **"RecursionError: maximum recursion Depth exceded"** al ejecutar el código?

La causa del error presentado antes es que al hacer una función recursiva esta se llama muchas veces, o explicado más a fondo, esta excede el límite que hay dentro del programa para llamar una función o usar recursividad, por lo tanto, el programa no puede continuar con la función y simplemente dejara de funcionar y mandara este error.

5) ¿Cuál es la causa por la que termina anormalmente el programa?

La causa inicial por la que el programa termina anormalmente es debido a que se excede el límite de recursión predeterminado de Python, lo que genera el error "RecursionError: maximum recursion Depth exceded". Sin embargo, después de ajustar el límite de recursión, el programa todavía termina abruptamente. Esto sugiere que, aunque el límite de recursión se incrementó, el programa aun no tenía suficiente espacio en la pila para manejar las llamadas recursivas extensas. El código del Segmento 10 aborda este problema al ajustar el tamaño de la pila con "threading.stack_size(67108864*2)", lo que asigna más espacio a la pila. Adicionalmente, el programa se ejecuta en un nuevo hilo con "thread = threading.Thread(target=menu_cycle)" y "thread.start()". Al hacer esto, el nuevo hilo utiliza el tamaño de pila ajustado, permitiendo que nuestro programa maneje llamadas recursivas más extensos sin agotar el espacio en la pila.

6) ¿Qué es lo que hacen las modificaciones del **main** al incluir el uso de **threading** en la ejecución del código?, utilice el código **¡Error! No se encuentra el origen de la referencia.** y la documentación oficial de Python para argumentar su respuesta.

Las modificaciones en el "main" que incluyen el uso de "threading" en la ejecución del código tienen dos objetivos principales:

1) **<u>Ajustar el tamaño de la pila para el hilo:</u>**
   “threading.stack_size(67108864*2)”
   Esta línea ajusta el tamaño de la pila predeterminado para los nuevos hilos creados. Al aumentar el tamaño de la pila a 128 MB, se permite que el programa maneje llamadas recursivas mucho más profundas que con el tamaño de pila predeterminado.

2) **<u>Ejecutar el ciclo del menú en un nuevo hilo:</u>**
   “thread = threading.Thread(target = menu_cycle
   thread.start”
   Estas líneas crean y comienzan un nuevo hilo que ejecutara la función “menu_cycle”. Dado que este hilo se crea después de ajustar el tamaño de la pila con “threading.stack_size()”, este hilo utiliza el nuevo tamaño de pila de 128 MB. Esto significa que cualquier llamada recursiva realizada dentro del “menu_cycle” podrá aprovechar este espacio adicional en la pila.

7) ¿Qué diferencias existen entre exceder el límite de recursión y la terminación anormal del programa?
Exceder el límite de recursión se refiere específicamente a cuando llamas una funcion recursivamente demasiadas veces. En Python existe un error llamado RecursionError para estos casos. Terminación anormal del programa incluye, pero no se limita a este error, ya que cualquier terminación del programa por cualquier error podría clasificarse como una terminación anormal del programa.