Aldana Smyna Medina Lostaunau (aldana.medina@udc.es)

Claudia Vidal Otero (claudia.votero@udc.es)

Manual de Usuario: Simulador gestión de catálogo de cursos

Al iniciar este programa, puedes cargar los cursos de las academias A y B desde dos archivos de texto, leer los ficheros de cursos A y B e insertarlos en árboles AVL, visualizar el resultado de realizar la operación "oferta agregada" o "oferta común" y visualizar diversas estadísticas sobre los árboles.

A continuación, tiene una guía paso a paso para comenzar a utilizar el catálogo:

1. Preparación del Archivo de Entrada

Antes de comenzar, asegúrate de tener un archivo de texto que contenga la información sobre los cursos que quieres subir al catálogo. El archivo debe seguir el siguiente formato:

Nombre,Duración(horas),Número de alumnos,Nivel,Idioma,Precio

Ejemplos:

Rust,60,10,A,Cas,15.00

C,120,30,A,Cas,8.00

Rust,60,10,A,Cas,15.00

Cada curso debe estar en una línea diferente, y, cada uno de sus atributos separado por coma.

El nivel se simboliza con las letras A, B o C y del idioma sólo pondremos las tres primeras letras.

2. Ejecución del Programa

Para ejecutar el programa, abre una terminal y navega hasta el directorio donde se encuentran los archivos del simulador (main.py, cursos.py), los archivos necesarios para el manejo de árboles AVL y los archivos de entrada. Luego, ejecuta el programa de la siguiente manera:

python main.py

A continuación, te saldrá en la terminal el siguiente menú:
--- Menú ---

1. Leer los ficheros de los cursos e insertarlos en árboles AVL
2. Operación 'oferta agregada' - cursos realizados por cada academia
3. Operación 'oferta común' - cursos realizados por ambas academias
4. Mostrar métricas
5. Salir

Para iniciar el programa deberás obligatoriamente seleccionar la opción 1, tras ello te saldrán los siguientes mensajes:

- Introduce el nombre del archivo txt de la academia A:ejA.txt
- Introduce el nombre del archivo txt de la academia B:ejB.txt

Deberás poner el nombre de los archivos que vayas a utilizar, como mostramos en el ejemplo con los archivos "ejA.txt" y "ejB.txt".

Descripción de las Fases de Desarrollo

- **Análisis de Requisitos:**

En esta fase, se identificaron los requisitos del proyecto basados en la especificación proporcionada. Se necesita gestionar un catálogo de academias con ciertas funcionalidades requeridas: la carga de datos desde dos archivos, la creación de árboles AVL, recorrer los árboles y crear dos clasificaciones "oferta agregada"— cursos realizados por cada academia—y como una "oferta común"— cursos realizados en ambas academias—, y la visualización de estas ofertas y métricas.

- **Arquitectura del Sistema:**

La arquitectura del sistema sigue un enfoque de programación orientada a objetos (OOP). Se utilizan árboles binarios AVL para almacenar y ordenar los cursos de manera eficiente que facilite la manipulación de datos en el catálogo de películas.

- **Diseño detallado:**

Al inicio, se importan tres módulos: el archivo que contiene la implementación de los árboles AVL, el archivo curso.py que comprende la clase Curso con sus atributos y getters y setters, y finalmente importamos pandas para calcular las métricas.

En main.py, se ha creado la clase SimulatorAcademias que implementa todas las funciones necesarias para las acciones que se piden. Empezamos creando las listas para las estadísticas (arbolB_data = [], arbolA_data = []) en donde se insertarán todos los cursos que le correspondan mediante el método **leer_cursos** que recorre el archivo dado y devuelve una instancia de su árbol AVL. Este archivo es pedido mediante un input en la función **execute_menu** que creamos para implementar el menú (creamos dos inputs, uno por cada academia) donde ofrecemos al usuario la elección de las distintas funciones, siendo necesario que elijan la primera opción (1. Leer los ficheros de los cursos e insertarlos en árboles AVL), si no se cumple, se visualizará por la pantalla un aviso.

Creamos los métodos **oferta_agregada** y **oferta_comun** para combinar cursos de dos árboles de diferentes academias. En **oferta_agregada**, fusionamos cursos con claves idénticas, seleccionando el de mayor beneficio, y renombramos cursos con nombres iguales pero claves diferentes. En **oferta_comun**, encontramos cursos presentes en ambos árboles, seleccionando el de mayor beneficio en caso de claves idénticas. Ambos métodos emplean la función **seleccionar_curso_mayor_beneficio**.

La función **metrics** calcula estadísticas sobre los cursos, utilizando la biblioteca 'pandas' para realizar estas operaciones, además, creamos una clase Metrics donde se almacenan y acceden a los atributos nombre, duración, número de alumnos, nivel y precio.

- **Pruebas y Depuración:**

Se hicieron pruebas exhaustivas para cada opción del menú y el programa. Se verificó que la carga de archivos funcionara correctamente y que los cursos se almacenaran y ordenaran según lo esperado. Se comprobó también el funcionamiento de las operaciones "oferta agregada" y "oferta común" y se verificó que la visualización de árboles y métricas proporcionara resultados precisos y coherentes. Además, se corrigieron errores y se depuró el código para garantizar su correcto funcionamiento en diferentes escenarios.

- **Documentación y Entrega:**

Primero redactamos la documentación del código, incluyendo comentarios explicativos y docstrings para cada método y clase. Luego creamos un manual de usuario que describe cómo utilizar el programa y una descripción detallada de las fases de desarrollo realizadas. Finalmente, entregamos el código fuente final junto con la documentación correspondiente.