Memoria de proyecto

Hugo Herrador Segade
h.herrador@udc.es

Pablo Montes Aldao
pablo.montes1@udc.es

3 de mayo de 2024

1. Introducción

El presente documento representa la memoria de la cuarta práctica de la asignatura Fundamentos de Programación II (GCED-FP2 614G020092324), desarrollada por Hugo Herrador Segade y Pablo Montes Aldao entre los días 22 de abril de 2024 y 3 de mayo del mismo año.

El programa Python desarrollado pretende abordar la gestión del catálogo de cursos de dos academias de idiomas, permitiendo generar un catálogo de "oferta agregada" y otro de oferta común, además de mostrar estadísticas sobre las academias (utilizando la librería pandas).

El programa funciona haciendo uso del TAD AVL, cuyo código se incluye en el directorio AVL.

2. Guía al usuario

El programa está diseñado para ser utilizado de forma sencilla y trata de acompañar al usuario a través de los menús para que obtenga lo que desea.

2.1. Requerimientos

Para ejecutar el programa se requiere:

- Un ordenador con un intérprete de Python (programa encargado de interpretar y ejecutar el código) instalado, versión 3.7 o más reciente
- Un gestor de paquetes y los paquetes indicados en el archivo requirements.txt

2.2. Mecanismo de ejecución del programa

En primer lugar, es importante señalar que tanto los archivos de código fuente (main.py, manager.py course.py) como los archivos de código del TAD AVL deben encontrarse en la misma carpeta, sin directorios anidados.

Para ejecutar el programa el usuario deberá llamar a main.py a través del intérprete de Python. En Linux, se puede hacer abriendo una terminal y ejecutando:

```
python3 main.py
```

Al ejecutar el programa, se crea un objeto de clase Manager sobre el que se aplica el método self.launch(). Al ejecutarse esté método, al usuario se le dará la opción de leer dos archivos de texto con datos de academias. Estes deben estar en el mismo formato que los ejemplos ejA.txt y ejB.txt. En caso contrario, se imprimirá un mensaje de error y se cerrará el programa.

Es importante resaltar que los ficheros a leer también se deben encontrar en el mismo directorio que los archivos de código Python. Alternativamente, se puede indicar como nombre de fichero la ruta al mismo.

Una vez leído el archivo, se crearán objetos de clase Course y se almacenarán en árbol AVL, que será un atributo de la clase Manager (academy_a o academy_b, según corresponda).

Después se llama al método self.main_menu(), que es el que gestiona el menú principal. Ahí dependiendo del input del usuario se llama a un método u otro para gestionar su petición. Después de satisfacerla, se vuelve a llamar a self.back_main() para volver al menú principal haciendo un efecto de espera. Veamos las opciones del usuario:

- Realizar la operación "oferta agregada" y visualizar el resultado. En caso de cursos iguales, se selecciona el curso con mayor beneficio (precio por hora y estudiante) y el número de estudiantes de los grupos fusionados se suma.
- Realizar la operación "oferta común" y visualizar el resultado. Se opera de la misma forma que en la opción anterior, seleccionando el curso de máximo beneficio en caso de haber varios iguales.
- Mostrar varias métricas (número medio de estudiantes por idioma, número medio de estudiantes por nivel e ingresos totales posibles), haciendo uso de la librería pandas. Para esto utilizamos el método self.show_stats() que crea un dataframe con la información de cada academia y calcula estadísticas sobre estos.

3. Fases de desarrollo

3.1. Fase I: Creación de la clase Course en el módulo course.py

En primer lugar, creamos la clase Course para representar cada una de los cursos. Contiene una serie de atributos comunes a todos los cursos (name, duration, enrolment, level, language, price, academy), que inicializaremos usando el método mágico __init__. Siguiendo la convención de *estilo Pythonic*, definiremos métodos de acceso y modificación para los atributos, utilizando los decoradores @property y @setter.

Además, redefinimos el método mágico __hash__(self) para obtener un identificador único para cada curso distinto y definimos también los métodos mágicos __eq__(self, other) y __gt__(self, other) para comparaciones entre objetos de la clase.

3.2. Fase II: Implementación de la clase Manager en manager.py

Una vez desarrollada la clase Film, comenzamos con la clase Manager, que se encargará de la gestión de los árboles. En esta clase incluimos los atributos y métodos necesarios para los procesos que queramos que se puedan realizar. Necesitaremos los siguientes atributos, que inicializaremos en el __init__:

- academy_a: Donde se almacenará un árbol con los cursos de la academia "A". Inicialmente None.
- academy_b: Donde se almacenará un árbol con los cursos de la academia "B". Inicialmente None.
- common: Árbol con los cursos comunes a ambas academia. Inicialmente es un árbol vacío
- aggregate: Árbol con la oferta agregada de ambas academia. Inicialmente es un árbol vacío
- done: Lista auxiliar para mantener registro de los cálculos que ya se han realizado. Inicialmente lista vacía.

Además, desarrollamos los siguientes métodos:

- read_file(self): Que permite leer el archivo, crear objetos de clase Course y almacenarlos en un árbol AVL.
- launch(self): Para gestionar el primer menú con el que tiene que interactuar el usuario cuando corre el programa. En este se le pide al usuario el nombre de los archivos que quiere leer.
- main_menu(self): Es el menú principal del programa. Permite al usuario decidir qué quiere hacer con los datos y llama al método que permita gestionar la petición.
- comm_offer(self): Crea un árbol con la "oferta común" (los cursos ofertados en ambas academias) y lo almacena en el atributo common. Si este árbol ya había sido creado antes llama al método self.warning()
- agg_offer(self): Crea un árbol con la "oferta común" (los cursos ofertados en ambas academias) y lo almacena en el atributo common. Si este árbol ya había sido creado antes llama al método self.warning()
- warning(self): Lanza un aviso al usuario si ya ha calculado el árbol que pide, y le pregunta si quiere volver a imprimirlo (sin calcular lo mismo de nuevo)
- back_main(self): Vuelve al menú principal con esperas, simulando una pantalla de carga

Por último, creamos un archivo main.py en el que se crea un objeto de la clase Manager y se le aplica el método self.launch(), para comenzar el programa.

3.3. Fase III: Cálculo de estadísticas con pandas

Ahora queremos desarrollar un método show_stats(self) que muestre diversas estadísticas sobre las academias.

Creamos variables data_a y data_b para almacenar listas con datos de los cursos de las academias A y B, respectivamente. Después convertimos dichas listas a dataframe para obtener:

- Número medio de estudiantes por idioma
- Número medio de estudiantes por nivel
- Ingresos totales posibles

4. Resultados y discusión

Se realizó un programa Python que permite al usuario, a partir de dos ficheros de texto con datos de academias: almacenar los datos en árboles AVL, crear árboles con la oferta agregada y la oferta común y mostrar estadísticas de las academias.

El menú es intuitivo, amigable con el usuario y trata los errores que pueden suceder en la ejecución, lanzando mensajes explicativos cuando es necesario.

No se detecta ningún error en el funcionamiento del programa tras realizar exhaustivas pruebas con ficheros de prueba creados, de entre 1 y 2000 cursos, con y sin repetidos.