Práctica 3. Árboles AVL

Sesiones de prácticas: 2

Objetivos

Implementar la clase `AVL<T>` utilizando **patrones de clase y excepciones**. Programa de prueba para comprobar su correcto funcionamiento.

Descripción de la EEDD

Implementar la clase `AVL<T>` para que tenga toda la funcionalidad de un árbol equilibrado AVL en memoria dinámica, tal y como se describe en la Lección 11, utilizando patrones de clase y excepciones. Los métodos a implementar serán los siguientes:

En concreto se usará:

- Constructor por defecto `AVL<T>()`
- Constructor copia `AVL<T>(const AVL<T>& origen)`.
- Operador de asignación (=)
- Rotación a derechas dado un nodo `rotDer(Nodo<T>* &nodo)`
- Rotación a izquierdas dado un nodo `rotIzq(Nodo<T>* &nodo)`
- Operación de inserción `bool inserta(T& dato)`
- Operación de búsqueda recursiva `T* buscaRec(T& dato)`
- Operación de búsqueda iterativa `T* buscaIt(T& dato)`
- Recorrido en inorden[1] `VDinamico<T*> recorreInorden()`
- Número de elementos del AVL `unsigned int numElementos()`
- Altura del AVL, `unsigned int altura()`
- Destructor correspondiente

Tanto el constructor de copia como el operador de asignación deben crear copias idénticas. El contador de números de elementos puede realizarse mediante un atributo contador o mediante un recorrido en O(n).

[1] Devuelve un vector dinámico de punteros a los elementos del AVL haciendo un recorrido en Inorden

Descripción de la práctica: crear un gestor de imágenes

En esta práctica seguiremos mejorando la funcionalidad del diseño de ImageBook con la incorporación de la clase *Usuario*. La nueva relación *ImageBook::users* se creará mediante un AVL instanciado del modo *AVL<Usuario>*, para lo cual se adjunta un fichero de usuarios (*usuarios.txt*). Estos usuarios coinciden con los ya existentes en el fichero "*images_v1.csv"* . Esto implica que el email del usuario ya no formará parte de la clase *Imagen*.

Un usuario tiene ahora una relación "a muchos" con la clase *Imagen* a través de la relación de asociación *Usuario::userImages*, implementada con un vector dinámico VDinamico<T>.

Para nutrir de datos el sistema en el constructor de ImageBook, se puede seguir la siguiente secuencia de pasos:

1. Leer el fichero *"etiquetas.txt"* y cargarlos en *ImageBook:labels* mediante una lista como en la *Práctica 2*.
2. Leer el fichero *"usuarios.txt"* y cargarlos en *ImageBook::users,* dejando vacía la estructura de datos de la asociación *Usuario::userImages.* Se irá rellenando según se lea el fichero de imágenes.
3. Llamar al constructor del vector dinámico con el número de imágenes máximo que tendrá el fichero (10.000). Es importante fijar el tamaño lógico desde el principio para que no crezca al doble y se pierdan las referencias de las imágenes que lleguen al vector.
4. Leer cada una de las imágenes del fichero *"images_v1.csv"* y para cada una de ellas:
    a. Crear un objeto Imagen con todos los atributos del fichero (menos etiqueta y email).
    b. Buscar en *ImageBook::labels* la primera etiqueta que aparece en el fichero y enlazar correctamente la asociación *Imagen::etiquetada*, tal y como se hizo en la Práctica 2.
    c. La nueva imagen creada se añade ahora al vector del atributo *ImageBook::images* y acto seguido obtenemos su dirección de memoria en dicho vector (se localiza en la última posición del vector).
    d. Buscar el email del usuario de la imagen el AVL (relación *ImageBook::users*) y obtener al usuario asociado.
    e. Añadir en la estructura de datos *Usuario::userImages* dicha dirección de memoria de la imagen.

Es importante que todos los enlaces anteriores se realicen correctamente, para ello se obtienen los punteros o referencias de los objetos, nunca las copias.

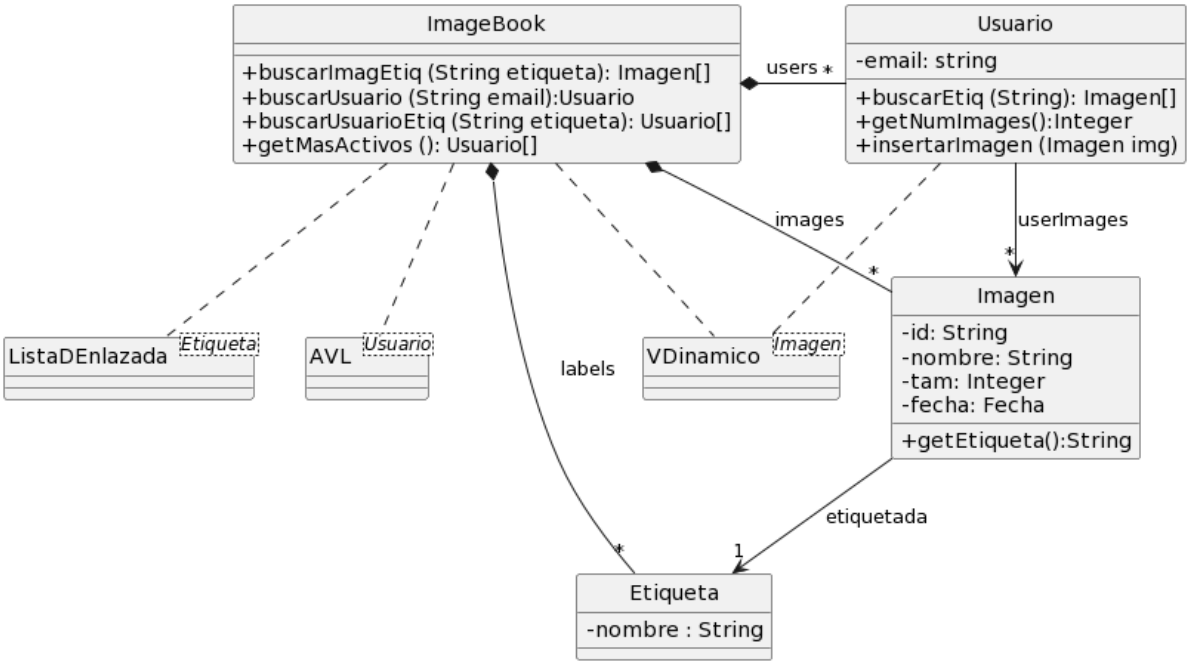
La funcionalidad de las clases es la siguiente considerando que nunca de devuelven objetos copia:

ImageBook

- *buscarImagEtiq()*, que devuelve en un vector dinámico (*Vdinamico<T>*) con todas las imágenes que tengan la etiqueta dada.
- *buscarUsuario()*, que devuelve el usuario con un email dado.
- *buscarUsuarioEtiq()*, que localiza a todos los usuarios que han colgado al menos una imagen con la etiqueta dada. Los usuarios se pueden obtener con la función AVL<T>::recorreInorden() y para cada uno de ellos llamar a Usuario::buscarEtiq().
- *getMasActivos()*, que devuelve el usuario (o usuarios en caso de empate) más activo en la red porque cuelga más imágenes que el resto. Para ello se puede llamar a *Usuario::getNumImages()*

Usuario

- *buscarEtiq()*, que devuelve las imágenes del usuario que coincidan con la etiqueta dada.
- *getNumImages()*, devuelve el número de imágenes que tiene el usuario colgadas.
- *Operadores relacionales: operator< () y operator> ()*, deben implementarse considerando el email como atributo de comparación.
- InsertarImagen (), que añade un nuevo dato en la relación Usuario::userImages



Programa de prueba: probar el gestor de imágenes

- Implementar la EEDD *AVL<T>* con la funcionalidad señalada arriba y de acuerdo con la especificación de la Lección 11. Hay operaciones como el recorrido en Inorden que se puede usar tal cual de la Lección 10.

- Probar la estructura, en una función independiente, instanciándola a AVL<unsigned int> con un millón de enteros aleatorios[2] en el rango [0, 1.000.000] y mostrar la altura del árbol por pantalla (puede que no todos se inserten si están repetidos).
- Instanciar la clase *ImageBook* según el diseño UML añadiendo los datos de los tres ficheros anteriormente y siguiendo el procedimiento anteriormente descrito.
- Buscar y mostrar la información de las imágenes de estos usuarios (si es que existen): eliza39@yahoo.com, betty95@hotmail.com, betty95@hotmail.com, victor6@gmail.com y manolete@gmail.com.
- Devolver y mostrar por pantalla todos aquellos usuarios que hayan publicado alguna imagen con la etiqueta "playa" y posteriormente los que hayan publicado con la etiqueta "comida".
- Devolver el/los usuarios más activos en la red porque hayan publicado más imágenes.
- **Para los que trabajan por parejas**:
    - Medir el tiempo total de ejecución de 1000 operaciones de búsqueda de valores aleatorios en el *AVL<unsigned int>* del segundo apartado.
    - Recorrer en Inorden dicho árbol e introducir en un vector dinámico *VDinamico<T>* los elementos que están en el rango [1000, 10.000].
    - Mostrar el tamaño del vector anterior.
    - Mostrar los 200 primeros números.

Estilo y requerimientos del código:

1. El código debe ser claro, tener un estilo definido y estar perfectamente indentado, para ello se pueden seguir algunos de los estilos preestablecidos para el lenguaje C++ (http://geosoft.no/development/cppstyle.html).

2. Deben comprobarse todas los posibles errores y situaciones de riesgo que puedan ocurrir (desbordamientos de memoria, parámetros con valores no válidos, etc.) y lanzar las excepciones correspondientes, siempre que tenga sentido. Leer el tutorial de excepciones disponible en el repositorio de la asignatura en docencia virtual.

3. Se valorará positivamente la calidad general del código: claridad, estilo, ausencia de redundancias, etc.

[2] Utilizar la función *std::rand()* https://en.cppreference.com/w/c/numeric/random/rand