Práctica 5. Tabla hash cerrada

Sesiones de prácticas: 2

Objetivos

Implementación y optimización de tablas de dispersión cerrada.

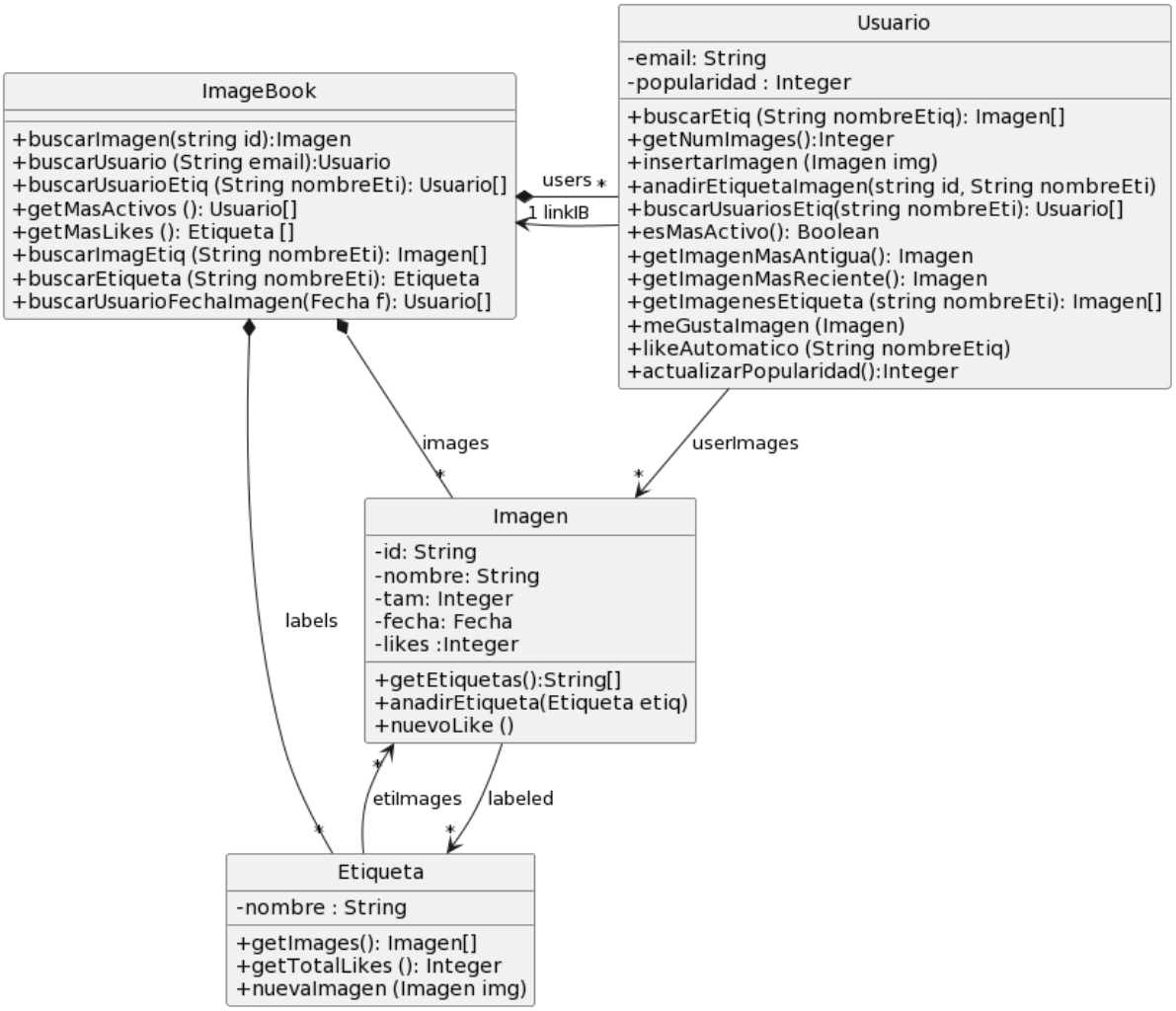
Descripción de la EEDD

En esta práctica se implementará una tabla hash de dispersión cerrada de imágenes, por lo que no se implementará esta vez mediante un template[1]. Su definición sigue esta especificación:

- *THashImagen::hash(unsigned long clave, int intento)*, función privada con la función de dispersión.
- *THashImagen::THashImagen(int maxElementos, float lambda=0.7)*. Constructor que construye una tabla garantizando que haya un factor de carga determinado al insertar todos los datos previstos.
- *THashImagen::THashImagen(THashImagen &thash)*. Constructor copia.
- *THashImagen::operator=(THashImagen &thash)*. Operador de asignación.
- *~THashImagen()*. Destructor.
- *bool THashImagen::insertar(unsigned long clave, Imagen &imagen)*, que inserte una nueva imagen en la tabla[2]. Como no se permiten repetidos, si se localiza la clave en la secuencia de exploración no se realizará la inserción, devolviendo falso para indicarlo.
- *Imagen * THashImagen::buscar(unsigned long clave, string &id)*, que busque una imagen a partir de su clave de dispersión numérica y devuelva un puntero a la imagen localizada (o *nullptr* si no se encuentra).
- *bool THashImagen::borrar(unsigned long clave, string &id)*, que borre la imagen de la tabla. Si el elemento no se encuentra devolverá el valor falso.
- *unsigned int THashImagen::numImages()*, que devuelva de forma eficiente el número de imágenes que contiene la tabla.

[1] Para facilitar la experimentación con diferentes funciones de dispersión, el usuario de la tabla podrá proporcionar el valor de dispersión numérico de las claves utilizando la función de dispersión que considere oportuno.
[2] IMPORTANTE: Al hacer la implementación consideraremos que la secuencia de exploración encontrará siempre la posición de inserción tras un número suficiente de iteraciones ya que vamos a trabajar con factores de carga adecuados. Ver ejercicio adicional por parejas para comprender cómo se garantiza esto independientemente del número de elementos que se inserten.

Se actualizará el diseño según el siguiente esquema UML.



Recordad que las claves en dispersión deben ser de tipo unsigned long, por lo que un string debe ser previamente convertido a este tipo mediante la función *djb2()* al llamar a los métodos de la clase.

Descripción de la práctica:

En esta práctica se va a aumentar la funcionalidad de la misma mediante la inclusión de **nuevas funciones y relaciones**. La clase *ImageBook*, al igual que en las prácticas anteriores, se encargará de la gestión del conjunto de imágenes y mantendrá todas sus relaciones. De igual forma, la clase *Usuario* también mantendrá sus relaciones tal y como estaban en la práctica anterior. Sin embargo, la clase *Etiqueta* introducirá una nueva relación con *Imagen*, además de las existentes con el resto de clases. Esta **nueva relación (*etiqImages*)** será una asociación y ofrece la posibilidad de poder obtener las imágenes que están vinculadas a una etiqueta concreta. Usar un contenedor std::list para implementar esta relación.

Para mejorar la búsqueda de las imágenes **modificaremos la composición *ImageBook::images*** para que utilice la tabla de dispersión implementada en vez del vector dinámico, adaptando los métodos de ImageBook que sean necesarios.

Además de estos cambios, se va a introducir el **concepto de "*like*" o "*me gusta*"** en las imágenes de los usuarios. Para ello, se va a añadir a la clase *Imagen* un nuevo atributo, *likes*, que va a representar el recuento de *likes* de una imagen. Al contar con esta nueva cualidad de las imágenes, el *Usuario* podrá dar *likes* a las imágenes de otros usuarios, así como también tendrá un nuevo atributo *"popularidad"*

que representará la popularidad del *Usuario* en este gestor de imágenes. La popularidad de un usuario hace referencia al recuento de *"likes"* que han tenido sus imágenes; cuantos más *"likes"* tienen las imágenes de un usuario, más popular es.
Las funcionalidades nuevas o que resultan afectadas de las clases son las siguientes considerando que los métodos nunca devuelven objetos copia:

ImageBook:

- *getMasLikes (): Etiqueta []* → obtiene las etiquetas con más likes.
- *buscarImagEtiq (String nombreEti): Imagen[]* → busca todas las imágenes que tienen la etiqueta dada. Esta operación se realizará a través de la relación etiImages.
- *buscarEtiqueta (String nombreEti): Etiqueta* → busca una etiqueta cuyo nombre se pasa por parámetro.

Imagen:

- *nuevoLike()* → permite dar *likes* a una imagen.
- *getLikes()*: Integer → obtiene el número de *likes* de una imagen.

Etiqueta:

- *getImages(): Imagen[]* → obtiene todas las imágenes vinculadas a esa etiqueta.
- *getTotalLikes(): Integer* → obtiene el total de likes que tiene esa etiqueta. Es decir, obtiene el total de likes global de todas las imágenes vinculadas a esa etiqueta.
- nuevaImagen(Imagen) → establece una asociación entre la imagen proporcionada y la etiqueta

Usuario:

- *getImagenesEtiqueta (string nombreEti): Imagen[]* → obtiene todas las imágenes con la etiqueta dada de todos los usuarios del sistema. Por ejemplo, todas las imágenes subidas con la etiqueta "gato".
- *meGustaImagen (Imagen)* → añadirá un like a la imagen de otro usuario que se le pase por parámetro. En ningún caso podrá darse like a sus propias imágenes.
- *likeAutomatico (String nombreEtiq)* → incrementará automáticamente en una unidad el valor de likes de todas las imágenes vinculadas a una etiqueta (ej. a todas las imágenes que tenga la etiqueta "gato").
- *actualizarPopularidad()* → actualiza la popularidad del usuario al número de likes que tiene el conjunto de sus imágenes. Se consigue sumando los likes de todos sus imágenes. La utilidad de este método es permitir la consulta eficiente de la popularidad de un usuario con el método *Usuario::getPopularidad()* y sería llamado periódicamente por el propio sistema.

Descripción de la práctica:

1ª Parte: Ajuste de la tabla

Antes de que la tabla deba ser utilizada, se debe entrenar convenientemente para determinar qué configuración es la más adecuada. Para ello se van a añadir nuevas funciones que ayuden a esta tarea:

- *unsigned int THashImagen::maxColisiones()*, que devuelve el número máximo de colisiones que se han producido en la operación de inserción más costosa realizada sobre la tabla.
- *unsigned int THashImagen::numMax10()*, que devuelve el número de veces que se superan 10 colisiones al intentar realizar la operación de inserción sobre la tabla de un dato.
- *unsigned int THashImagen::promedioColisiones()*, que devuelve el promedio de colisiones por operación de inserción realizada sobre la tabla.

- *float THashImagen::factorCarga()*, que devuelve el factor de carga de la tabla de dispersión.
- *unsigned int THashImagen::tamTabla()*, que devuelve el tamaño de la tabla de dispersión.
- *void ImageBook::mostrarEstadoTablaImagenes()* que muestra por pantalla los diferentes parámetros anteriores de la tabla interna de imágenes. Usar el método en main después de llamar al constructor de ImageBook cuando haya cargado todos los ficheros de datos.

Ayudándose de estas funciones, se debe completar una tabla (en formato markdown[3]) que contenga los siguientes valores: máximo de colisiones, factor de carga y promedio de colisiones con **tres funciones hash** y con **dos tamaños de tabla diferentes** (considerando factores de carga $\boldsymbol{\lambda}$ de 0,65 y de 0,68 respectivamente). Para determinar el tamaño de la tabla, hay que obtener el siguiente **número primo** después de aplicar el $\boldsymbol{\lambda}$ al tamaño de los datos.

Para simplificar el **proceso de ajuste de la tabla**, una vez sustituido el vector de imágenes por la tabla en ImageBook, y realizados los cambios en la tabla para cada experimento, en main utilizar únicamente el constructor de ImageBook para precargar los datos de los ficheros y, posteriormente, mostrar el estado interno de la tabla con el método *ImageBook::mostrarEstadoTablaImagenes()*.

Se probarán una función de dispersión cuadrática y dos con dispersión doble, que debéis elegir libremente intentando que sean novedosas. En total salen 6 combinaciones posibles. En base a estos resultados, se elegirá la mejor configuración para balancear el tamaño de la tabla y las colisiones producidas. Las funciones de exploración descartadas deben aparecer comentadas o sin usar en la clase tabla de dispersión. **El fichero markdown a utilizar está disponible junto a este enunciado con el nombre *analisis_Thash.md* y debe incluirse completado con los resultados obtenidos en el contenido del proyecto**. Elegir de forma justificada la mejor configuración de la tabla en base al estudio anterior para realizar la segunda parte del ejercicio.

2ª parte: Programa de prueba

Una vez ajustada la tabla, modificar el programa de prueba con las siguientes indicaciones:

- Implementar las clases anteriores de acuerdo al diagrama UML.
- Instanciar el mapa con objetos de tipo *Usuario* con el fichero adjunto. La clase *ImageBook* toma el nombre del fichero en el constructor para la lectura del mismo.
- Instanciar la lista con objetos de tipo *Etiqueta* con el fichero adjunto. La clase *ImageBook* toma el nombre del fichero en el constructor para la lectura del mismo
- Sustituir el vector de imágenes por la tabla hash con objetos de tipo *Imagen* con el fichero adjunto. La clase *ImageBook* toma el nombre del fichero en el constructor para la lectura del mismo. En esta práctica se almacenarán todas las etiquetas asociadas a la imagen. Además, será necesario inicializar el número de *"likes"* de las imágenes a los tres últimos dígitos del id de la imagen.
    - En esta práctica se tendrá que vincular también cada imagen a sus respectivas etiquetas.
- Mostrar el factor de carga de la tabla junto al tamaño de la misma.
- El usuario kenny_ohara73@yahoo.com quiere darle *like* a la última imagen que ha subido el usuario magdalen_upton99@gmail.com. Encontrar la imagen, mostrar sus datos y darle *like*. Volver a mostrar los datos de la imagen.

[3] Los ficheros markdown, con extensión md, utilizan caracteres especiales para dar formato al contenido. Clion incorpora un editor integrado de contenidos en este formato

- El usuario beau1@hotmail.com quiere darle *like* a la imagen con id 32477162. Localizarla y darle *like.*
- Darle *like* a todas las imágenes con la etiqueta “gato”.
- Se quiere saber qué *etiquetas* son más influyentes. Para ello, obtener el top 5 de etiquetas con más likes.
- Se quiere saber qué usuarios son más populares. Para ello, mostrar el top 3 de usuarios más populares.
- Obtener el número de *likes* de la etiqueta "pantalla".
- Buscar y eliminar la imagen con id 58540348.
- Obtener el número de *likes* de la etiqueta "pantalla".
- Comprobar que no está la imagen 58540348 tras el borrado mediante una búsqueda e insertar dicha imagen de nuevo.
- Mostrar el número de colisiones máximo que se han producido al volver a insertar la imagen.

Nota: Para los que trabajan en parejas:

- Implementar *void THashImagen::redispersar(unsigned tam)*, que redispersa la tabla a un nuevo tamaño *tam.*
- Modificar el método insertar de la tabla de dispersión para que cuando se detecte que el factor de carga supera un lambda 0,68 lance el método redispersar a un tamaño de la tabla un 30% más grande.
- Probar el comportamiento del método añadiendo 5 imágenes nuevas a ImageBook[4] y mostrando después el estado de la tabla

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.

[4] Añadir y utilizar para ello un método *ImageBook::nuevaImagen(const Imagen &img)* que añada al sistema una copia de la imagen suministrada