Práctica 5. Tabla hash

Sesiones de prácticas: 2

Objetivos

Implementación y optimización de tablas de dispersión cerrada.

Descripción de la EEDD

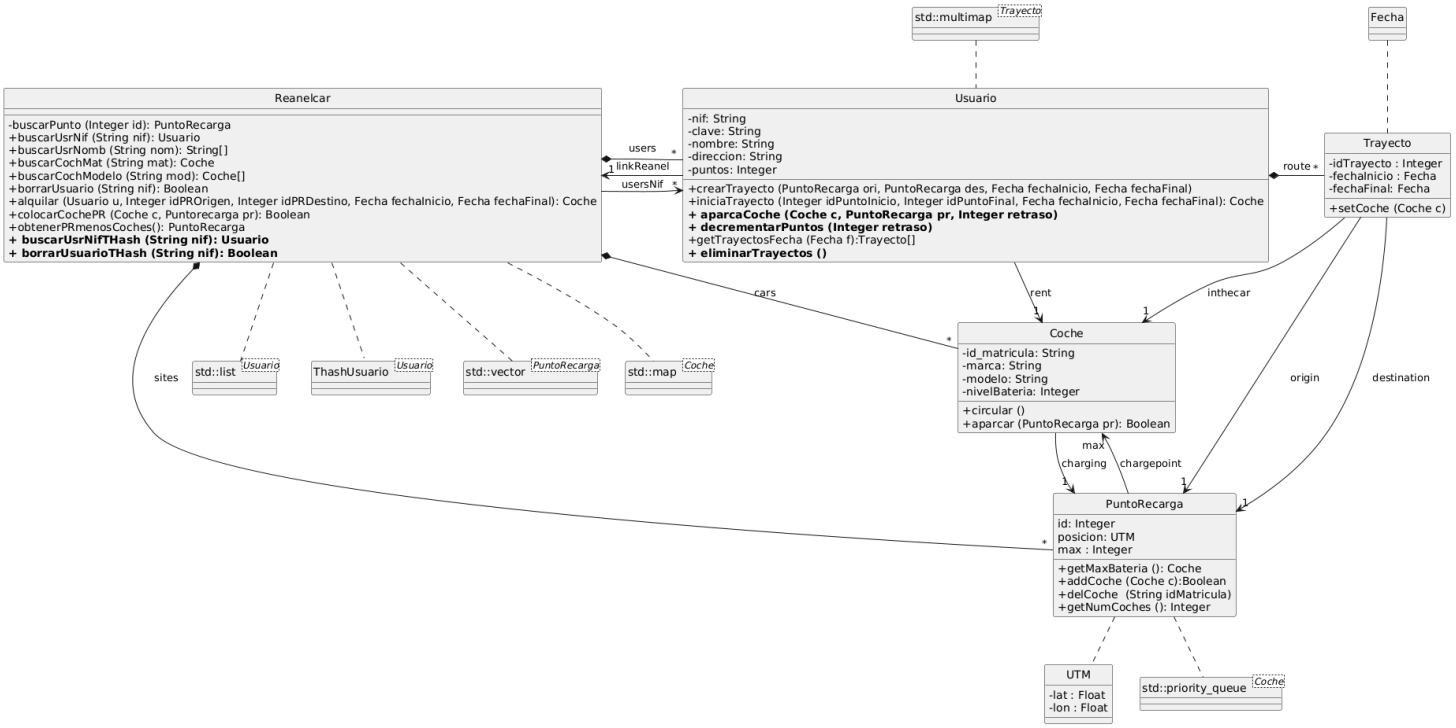
En esta práctica se implementará una tabla hash de dispersión cerrada de *Usuarios*, por lo que **no se implementará esta vez mediante un template**. Su definición sigue esta especificación:

- *ThashUsuario::hash(unsigned long clave, int intento)*, función privada con la función de dispersión.
- *ThashUsuario::ThashUsuario(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.
- *ThashUsuario::ThashUsuario(ThashUsuario &thash)*. Constructor copia.
- *ThashUsuario::operator=(ThashUsuario &thash)*. Operador de asignación.
- *~ThashUsuario()*. Destructor.
- *bool ThashUsuario::insertar(unsigned long clave, string &clave, Ususario &usuario)*, que inserte un nuevo usuario en la tabla[1]. Como no se permiten repetidos, si se localiza la clave en la secuencia de exploración no se realizará la inserción, devolviendo falso para indicarlo.
- *Usuario* ThashUsuario::buscar(unsigned long clave, string &clave)*, que busque un usuario a partir de su clave de dispersión numérica y devuelva un puntero al usuario localizado (o *nullptr* si no se encuentra).
- *bool ThashUsuario::borrar(unsigned long clave, string &clave)*, que borre el usuario de la tabla. Si el elemento no se encuentra devolverá el valor falso.
- *unsigned int ThashUsuario::numElementos()*, que devuelva de forma eficiente el número de elementos que contiene la tabla.

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.

[1] 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.



Descripción de la práctica

En esta práctica se mantendrá la funcionalidad de *Reanelcar* pero optimizaremos los contenedores existentes. La implementación de estas mejoras tiene como objetivo principal aumentar la eficiencia en la gestión de datos en *Reanelcar*. Una de las modificaciones será añadir una relación de asociación entre *Reanelcar* y *Usuario* (*Reanelcar::usersNif*), utilizando una tabla hash (***ThashUsuario***) para mejorar la eficiencia de las búsquedas. Además, cambiaremos la relación existente entre *PuntoRecarga* y *Coche* (*PuntoRecarga::chargepoint*) transformando el multimapa (std::multimap<int, Coche*>) por una cola con prioridad (***std::priority_queue<Coche*>)*** .

Resumiendo, los contenedores a usar para las seis relaciones "uno a muchos" son:

- ReanelCar::cars → std::map<string, Coche> con clave la matrícula
- ReanelCar::users → std::list<Usuario>
- ReanelCar::sites → std::vector<PuntoRecarga>
- **Reanelcar::usersNif → ThashUsuario**
- **PuntoRecarga::chargepoint → std::priority_queue<Coche*>** con prioridad el nivel de batería
- Usuario::route → std::multimap<Fecha, Trayecto> con clave la fecha de inicio

Debemos tener claro que el objetivo de la tabla hash es mejorar el rendimiento de las búsquedas, así que para ello haremos lo siguiente. Se buscará en la lista de usuarios un conjunto de usuarios de forma

secuencial, cuyo nombre comenzará por una letra dada, y se almacenarán sus nifs en un vector. A continuación, realizaremos las búsquedas de esos usuarios en la tabla hash a partir de sus claves. También podremos añadir y eliminar usuarios en la tabla hash y estudiaremos las colisiones que se producen.

En esta práctica se ha añadido a la clase *Usuario* el atributo *puntos*, que estará inicializado a 100, y se irá decrementando cada vez que el usuario no entregue el coche en la fecha indicada cuando hizo el alquiler. La penalización será de 2 puntos por cada hora de retraso. Para ello, se modifica el método *Usuario::aparcaCoche()* en el que se añade el número de horas de retraso que se entrega el coche. El método *Usuario::decrementarPuntos()* decrementa los puntos correspondientes.

La nueva funcionalidad de las clases es la siguiente considerando que los métodos nunca devuelven objetos copia de los objetos gestionados por el sistema:

Reanelcar

- *Reanelcar::buscarUsrNifTHash(String nif): Usuario*, dado el nif de un usuario, devuelve el usuario con dicho nif.

- *Reanelcar::borrarUsuarioTHash(String nif):* dado el nif de un usuario, elimina dicho usuario y su información relacionada en el sistema: trayectos y coches alquilados.

Usuario

- *Usuario::eliminarTrayectos()*: eliminar todos los trayectos de un usuario.
- *Usuario:.decrementarPuntos(Integer retraso)*: decrementa los puntos correspondientes dependiendo del retraso en horas producido al aparcar el coche en el punto de recarga destino.

Programa de prueba 1: 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 ThashUsuario::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 ThashUsuario::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 ThashUsuario::promedioColisiones()*, que devuelve el promedio de colisiones por operación de inserción realizada sobre la tabla.
- *float ThashUsuario::factorCarga()*, que devuelve el factor de carga de la tabla de dispersión.
- *unsigned int ThashUsuario::tamTabla()*, que devuelve el tamaño de la tabla de dispersión.
- *void Reanelcar::mostrarEstadoTabla()* que muestra por pantalla los diferentes parámetros anteriores de la tabla interna de usuarios. Usar el método en main después de llamar al constructor de *Reanelcar* cuando haya cargado todos los ficheros de datos.

Ayudándose de estas funciones, se debe completar una tabla (en formato markdown[2]) que contenga los siguientes valores: máximo de colisiones, factor de carga y promedio de colisiones con **tres funciones hash** (una de dispersión cuadrática y dos de dispersión doble) 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**, en main utilizar únicamente el constructor de *Reanelcar* para precargar los datos de los ficheros y, posteriormente, mostrar el estado interno de la tabla con el método *Reanelcar::mostrarEstadoTabla()*.

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 (la justificación debe venir explicada).

Pasos a seguir:

1. Implementar la clase *THashUsuario*.
2. Elección de tres funciones hash (una cuadrática y dos de dispersión doble).
3. Cálculo del tamaño de la tabla utilizando el siguiente número primo después de aplicar el $\boldsymbol{\lambda}$ al tamaño de los datos.
4. Mostrar el estado interno de la tabla con el método *Reanelcar::mostrarEstadoTabla()*
5. Completar una tabla en formato markdown con valores como máximo de colisiones, factor de carga y promedio de colisiones para las seis combinaciones posibles.
6. Elegir la mejor configuración de la tabla en base al estudio para la segunda parte del ejercicio y documentarlo.

A continuación, se va a realizar una **prueba de rendimiento** para evaluar la eficiencia de la búsqueda de datos. La prueba consistirá en realizar una búsqueda masiva de datos de usuarios utilizando la nueva implementación de la tabla hash. Para ello, se buscarán (de forma secuencial) todos los usuarios que comienzan por W y se introducirán sus *nifs* en un vector. A partir de esos nifs se buscará su clave correspondiente en la tabla hash, usando el método *Reanelcar::buscarUsrNifTHash(String nif)*.

Pasos a seguir:

1. Implementar la prueba de rendimiento de la tabla hash.
2. Obtener los usuarios que comiencen por “W” de Reanelcar y obtener sus nifs.
3. Realizar las búsquedas correspondientes de usuarios en Reanelcar (almacenados en la tabla hash ThashUsuario) y medir los tiempos.
4. Comparar los resultados con la búsqueda de los mismos Usuarios mediante la lista y documentar la diferencia de tiempos.
5. Documentar y analizar los resultados obtenidos durante la prueba añadiéndolos al final del informe analisis_THash.md.

---

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

Programa de prueba 2

Crear un programa de prueba con las siguientes indicaciones:

1. Instanciar *Reanelcar* con los datos de los ficheros.
2. Mostrar el factor de carga de la tabla junto al tamaño de la misma. Mostrar el número de colisiones máximo que se han producido al insertar en la tabla.
3. Buscar todos los usuarios que comienzan por "W". Todos estos usuarios alquilan un coche. Para ello, se seleccionará el PuntoRecarga de forma secuencial, comenzando en el primer PuntoRecarga y terminando en el 50. Tras hacer esa asignación, si fuera necesario, se volvería a comenzar por el primero cíclicamente hasta asignar todos. Si algún punto de recarga se queda sin espacio libre, se pasaría al siguiente. El PuntoRecarga destino será el punto de recarga con id siguiente al punto de recarga de origen. Para la creación del trayecto, se inicializará el *idTrayecto* de forma incremental (el valor depende del número de trayectos realizados por ese usuario) y la fecha de inicio será el día de 12/11/2024. La fecha de fin se calculará sumando 1 ó 2 días de forma aleatoria a la fecha actual. Mostrar los 10 primeros usuarios con todos sus datos, incluido el trayecto y coche alquilado.
4. De los usuarios que han alquilado un coche, los que empiezan por "Wi" van a aparcar el coche con un retraso de 2h. Coger el PuntoRecarga de destino del trayecto creado anteriormente. Mostrar los trayectos de los 10 primeros usuarios con todos sus datos (origen, destino, coche alquilado y fecha), así como los puntos acumulados (una vez han sido penalizados).
5. Buscar en la tabla hash el usuario con nif "84538382N" y mostrar sus datos así como sus trayectos (si los tiene).
6. Eliminar el usuario con nif "84538382N". Volver a buscar el usuario borrado y, si no se encuentra, volver a insertarlo en el sistema. Mostrar ahora sus trayectos (no debería tener). Mostrar el estado de la tabla para comprobar el número de colisiones máximo que se han producido al volver a insertar el usuario. Tened en cuenta que al eliminar un usuario hay que borrar también sus trayectos y si tiene un coche alquilado en ese momento hay que quitar la asociación entre el usuario y coche.
7. Eliminar todos los usuarios que comienzan por "Wa". Comprobar el número de usuarios antes y después de realizar esta operación y mostrarlo por pantalla, además del estado de la tabla.
8. Todos los usuarios que comienzan por "Wi" van a alquilar de nuevo un coche (como se ha especificado en el apartado 3). Mostrar los 10 primeros usuarios con todos sus datos, incluido el trayecto y coche alquilado.
9. De los usuarios que han alquilado un coche, los que empiezan por "Wil" van a aparcar el coche con un retraso de 4h. Coger el *PuntoRecarga* de destino del trayecto creado anteriormente. Mostrar los trayectos de todos los usuarios que han alquilado un coche, junto con los datos (origen, destino, coche alquilado y fecha), así como los puntos acumulados.

Nota: Para los que trabajan en parejas:

- Implementar *void ThashUsuario::redispersar(unsigned tam)*, que redispersa la tabla a un nuevo tamaño.
- Modificar el método insertar de la tabla de dispersión para que cuando se detecte que el factor de carga supera un lambda determinado lance el método redispersar a un tamaño de la tabla un 30% más grande. Modificar también el método *Reanelcar::muestraEstadoTabla()* para que también muestre el número de redispersiones que han ocurrido en la tabla desde su creación.
- Forzar a que se realice la redispersión de la tabla tras completar todos los apartados de la práctica bajando el valor de lambda.

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.