Práctica 5. Tablas Hash

Sesiones de prácticas: 2

ATENCIÓN. Esta práctica y sucesivas deben ser realizadas por todos los alumnos.

Objetivos

Implementación y optimización de tablas de dispersión cerrada.

Descripción de la EEDD

Como la empresa tiene un conjunto bastante fidelizado de clientes, se ha propuesto cambiar el diseño para que las búsquedas por cliente sean aún más eficientes. Para ello se ha pensado en utilizar una tabla de dispersión en la relación de EcocityMoto con Cliente. Para ello se va a utilizar una tabla hash cerrada que contemple además de la búsqueda tanto nuevas inserciones como borrados de clientes.

Recordad que las claves en dispersión deben ser de tipo unsigned long, por lo que un DNI debe ser previamente convertido a este tipo mediante la función djb2(). Es conveniente usar siempre el mismo formato de DNI con respecto a mayúsculas/minúsculas.

La tabla de dispersión no se implementará esta vez mediante un template, su definición sigue esta especificación:

- `THashCliente::hash(unsigned long clave, int intento)`, función privada con la función de dispersión
- `THashCliente::THashCliente(tamTabla)`
- `bool THashCliente::insertar(unsigned long clave, string &dni, Cliente &cli)`, que inserte un nuevo cliente en la tabla. No se permiten repetidos.
- `bool THashCliente::buscar(unsigned long clave, string &dni, Cliente &*cli)`, que busque un dato a partir de su clave numérica y devuelva el objeto Cliente a través de un puntero.
- `bool THashCliente::borrar(unsigned long clave, string &dni)`, que borre el cliente de la tabla.
- `unsigned int THashCliente::numClientes()`, que devuelva el número de clientes.

- `void THashCliente::redispersar(unsigned tama)`, que redispersa la tabla a un nuevo tamaño

Programa de prueba 1:

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:

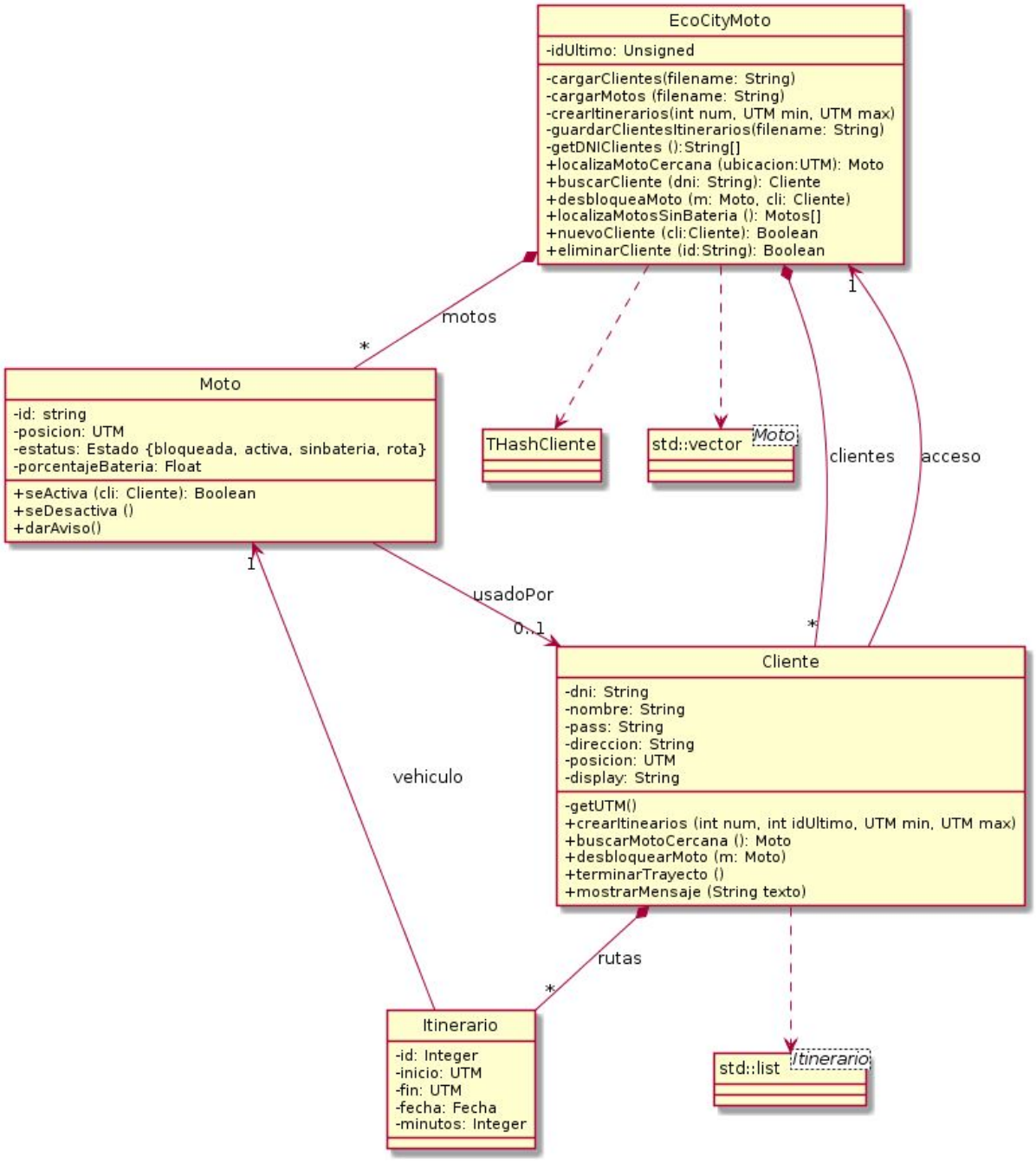
- `unsigned int THashCliente::maxColisiones()`, que devuelva el número máximo de colisiones que se han producido en la operación de inserción más costosa realizada sobre la tabla.
- `unsigned int THashCliente::promedioColisiones()`, que devuelva el promedio de colisiones por operación de inserción realizada sobre la tabla.
- `float THashCliente::factorCarga()`, que devuelva el factor de carga de la tabla de dispersión.
- `unsigned int THashCliente::tamaTabla()`, que devuelva el tamaño de la tabla de dispersión.

Ayudándose de estas funciones, se debe configurar una tabla (en word o similar) que rellene estos valores de máximo de colisiones, factor de carga y promedio de colisiones con tres funciones hash y con dos tamaños de tabla diferentes considerando un factor de carga $\boldsymbol{\lambda}$ >= 0.6. Se probará una función de dispersión cuadrática y dos con dispersión doble. 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. El fichero word debe llamarse *analisis_Thash* y debe subirse junto al proyecto.

Programa de prueba 2:

Una vez se tenga la configuración deseada de tabla hash, se modifica la implementación de la clase EcocityMoto con Cliente, con una funcionalidad similar a la práctica anterior según indica el siguiente UML. Se añade lo siguiente:

- un display en la aplicación móvil del cliente que indica en todo momento el estado de la moto, por ejemplo si está activa o si se ha quedado sin batería.



Nota: hay que tener en cuenta que cuando se realiza un nuevo itinerario, hay que actualizar la posición de la moto, con la posición final del recorrido.

Como prueba del ejercicio hay que realizar las siguientes operaciones:

1. Añadir un nuevo cliente que no exista previamente con coordenadas en Jaén,

rango (latitud, longitud): (37, 3) - (38, 4) .

3. Mostrar el número de colisiones que se han producido al insertarlo.

4. Localizar el cliente anterior en la empresa dado su DNI y mostrar toda su información por pantalla.

5. Buscar la moto más cercana al cliente anterior (que se pueda utilizar) y mostrar la información de la moto por pantalla..

6. Realizar un itinerario (dentro de la provincia de Jaén) con la moto localizada con una duración válida para la carga de batería de la moto. Actualizar el estado de la moto al finalizar el recorrido, atendiendo a la carga de batería.

7. Mostrar en el display el estado de la moto al inicio y fin del recorrido mediante el método *MostrarMensaje(String: texto)*.

8. Borrar el cliente anterior

9. Insertar de nuevo el mismo cliente (de esta forma comprobamos si se eliminó correctamente en el punto anterior).

10. Eliminar los primeros 1000 clientes del fichero, y cuando el factor de carga baje de 0.6, aplicar redispersión.

11. Volver a crear el fichero de itinerarios como en la Práctica 4, para recorrer todos los clientes, obtener todos los DNIs con EcoCityMoto::getDNIClientes() y luego encontrar todos ellos en la tabla hash.

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.