Práctica 5. Tablas Hash

Sesiones de prácticas: 2

Objetivos

Implementación y optimización de tablas de dispersión cerrada.

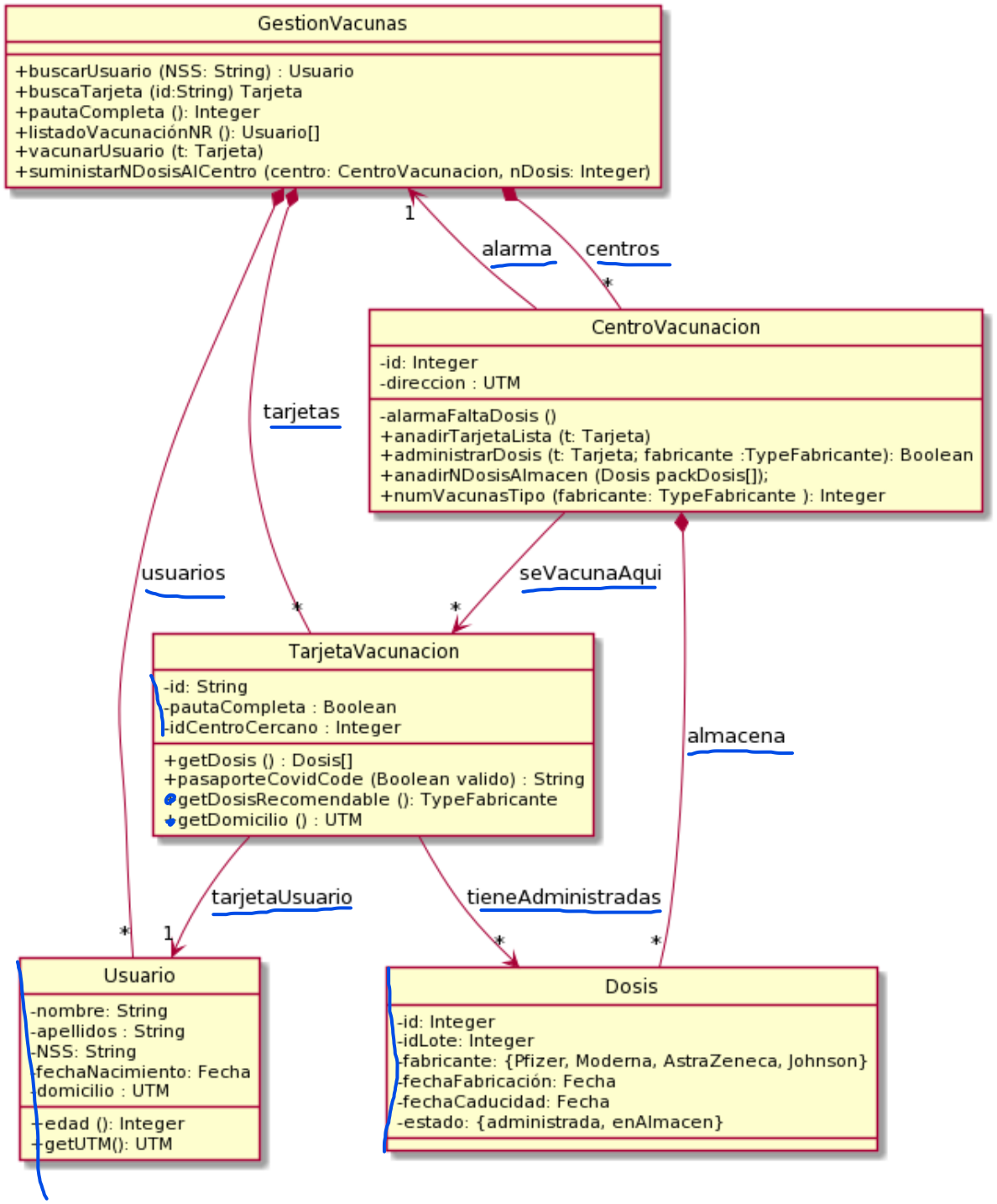
Descripción de la EEDD

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

- THashTarjetaVacunacion::hash(unsigned long clave, int intento), función privada con la función de dispersión.
- THashTarjetaVacunacion::THashTarjetaVacunacion(int tamTabla). Constructor que construye una tabla dado un tamaño predefinido.
- THashTarjetaVacunacion::THashTarjetaVacunacion(THashTarjetaVacunacion &thash). Constructor copia.
- THashTarjetaVacunacion::operator=(THashTarjetaVacunacion &thash).
- ~THashTarjetaVacunacion().
- bool THashTarjetaVacunacion::insertar(unsigned long clave, TarjetaVacunacion &pal), que inserte una nueva tarjeta de vacunación en la tabla. No se permiten repetidos por tanto, es necesario buscar antes de insertar.
- bool THashTarjetaVacunacion::buscar(unsigned long clave, string &id, TarjetaVacunacion &*pal), que busque un dato a partir de su clave numérica y devuelva el objeto TarjetaVacunacion a través de un puntero. Recordad que hay que comprobar que *id* coincide con la clave buscada por eso se pasa el parámetro *id* por la cabecera.
- bool THashTarjetaVacunacion::borrar(unsigned long clave, string &id), que borre la tarjeta de la tabla. Recordad que hay que comprobar que *id* coincide con la clave buscada antes de borrar.

- unsigned int THashTarjetaVacunacion::numTarjetas(), que devuelva el número de tarjetas de vacunación que contiene el diccionario.

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().

Descripción de la práctica:

La clase ***GestionVacunas*** se encargará de las vacunaciones de los usuarios a través del método *GestionVacunas::vacunarUsuario()*. El procedimiento para realizar dicha vacunación será en primer lugar localizar la tarjeta de vacunación del usuario y, seguidamente, a través de la misma, obtener el centro de vacunación más cercano al usuario[1] y se le asignará la tarjeta de vacunación a dicho centro de vacunación. El usuario sólo será vacunado si su tarjeta de vacunación está registrada en un centro de vacunación. Posteriormente, se llamará al método *CentroVacunación::administrarDosis()* donde se indica el tipo de dosis que se va a administrar al usuario a través de su tarjeta de vacunación. Si el usuario se encuentra registrado en el centro de vacunación correspondiente, esto es, que en el CentroVacunacion se haya añadido la tarjeta de vacunación del usuario (asociación *seVacunaAquí*), entonces se procederá a su vacunación. Tras realizar dicha vacunación, se eliminará la vinculación de la tarjeta de vacunación con el centro para que no pueda vacunarse sin permiso, y se devolverá true. En el caso de que el usuario no se encuentre registrado en el centro de vacunación, se devolverá false. Si el centro de vacunación se quedara sin dosis lo notificará a *GestionVacunas* mediante el método *CentroVacunacion::alarmaFaltaDosis()*, que llamará a *GestionVacunas::suministarNDosisCentro()* que a su vez envía las 100 dosis libres siguientes al centro de vacunación correspondiente a través del método CentroVacunacion*::anadirNDosisAlmacen()* que se usarán para vacunar a los usuarios a través de su tarjeta de vacunación de forma inmediata. La función *GestionVacunas::suministarNDosisCentro()* también puede usarse para inicializar las dosis en el sistema.

La clase ***CentroVacunacion*** representa los lugares donde los usuarios serán vacunados. Tiene una posición en el mapa dada por su posición UTM. **Cada centro tiene asociado una lista de tarjetas de vacunación de los usuarios registrados que serán los únicos que podrán ser vacunados en ese centro**. Las tarjetas de vacunación son registradas en el centro a través del método *CentroVacunacion::anadirTarjetaLista()*. Además, se podrá comprobar el número de dosis de cada tipo que quedan disponibles en el almacén de cada centro mediante el método *CentroVacunacion::numVacunasTipo()*.

La clase ***TarjetaVacunacion*** representa las tarjetas de vacunación de los usuarios, estando cada una de ellas asociada a un usuario concreto. La tarjeta tendrá un identificador único (id) generado mediante la concatenación del NSS del usuario y del nombre de este. Además, la tarjeta de vacunación nos informará de la dosis recomendable para ese usuario mediante el método *TarjetaVacunacion::getDosisRecomendable()*. Otra de las funcionalidades de la tarjeta de vacunación será la de generar un pasaporte covid mediante el método *TarjetaVacunacion::pasaporteCovidCode()*. Dicho método se encargará de, mediante una función hash[2] aplicada a la combinación del id de la tarjeta de vacunación junto al fabricante

---

[1] La distancia entre dos coordenadas UTM (un struct con dos campos de tipo float, latitud y longitud) se calcula como la distancia entre dos puntos https://es.wikipedia.org/wiki/Distancia

[2] Para la aplicación de la función hash anteriormente mencionada se usará un generador de hash SHA256 para *C++* (https://github.com/okdshin/PicoSHA2) incluyendo el fichero picosha2.h en nuestro proyecto.

de la primera dosis suministrada y el número de dosis, generar el código del pasaporte covid en caso de que la pauta de vacunación esté completa, retornando dicho código y true si se generó al tener la pauta completa o un código nulo y false en caso contrario.

En cuanto a las estructuras de datos, en la clase *GestionVacunas* los usuarios se almacenarán en un *Map<String, Usuario>* cuya clave vendrá determinada por el NSS del usuario. Las tarjetas de vacunación se almacenarán en la clase *GestionVacunas* en un *THashTarjetaVacunacion*. Los centros se almacenarán en un *Vector<Centro>*. En la clase *CentroVacunacion* las tarjetas de vacunación registradas se almacenarán en una *List<TarjetaVacunacion*>* y las dosis en un *Multimap<String, Dosis>* cuyas claves serán los fabricantes de las dosis. En la clase *TarjetaVacunacion*, las dosis administradas serán almacenadas en un *Vector<Dosis*>*. Obsérvese que las asociaciones son forzosamente definidas con puntero y las composiciones no tienen porqué, resultando más cómodo hacerlas sin puntero.

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 THashTarjetaVacunacion::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 THashTarjetaVacunacion::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 THashTarjetaVacunacion::promedioColisiones(), que devuelve el promedio de colisiones por operación de inserción realizada sobre la tabla.
- float THashTarjetaVacunacion::factorCarga(), que devuelve el factor de carga de la tabla de dispersión.
- unsigned int THashTarjetaVacunacion::tamTabla(), que devuelve 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 $\boldsymbol{\lambda}$ de 0,65 y de 0,68. 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.

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:

Crear un programa de prueba con las siguientes indicaciones:

- Implementar las clases anteriores de acuerdo al diagrama UML.
- Al igual que en la práctica anterior, instanciar el vector con objetos de tipo *CentroVacunacion* con el fichero adjunto. La clase *GestionVacunas* toma el nombre del fichero en el constructor como en la práctica anterior.
- Al igual que en la práctica anterior, instanciar el mapa con objetos de tipo *Dosis* de cada centro tomando el orden de lectura del fichero de dosis adjunto. *GestionVacunas* toma el nombre del fichero en el constructor como en la práctica anterior. Al primer centro de vacunación se le asignarán las primeras 8.000 dosis, al segundo las siguientes 8.200, al tercero las siguientes 8.500, al cuarto las siguientes 5000 y al último las 50 siguientes. Tened en cuenta que quedan 250 dosis sin repartir que se emplearán para proveer a los centros que se queden sin dosis.
- Al igual que en la práctica anterior, instanciar el mapa con objetos de tipo *Usuario* con el fichero adjunto. La clase *GestionVacunas* toma el nombre del fichero en el constructor como en la práctica anterior.
- Generar las tarjetas de vacunación de cada uno de los usuarios e instanciar la tabla hash con objetos de tipo TarjetaVacunacion. La clase *GestionVacunas* será la encargada de crear las tarjetas de vacunación.
- Mostrar el factor de carga de la tabla junto al tamaño de la misma.
- Eliminar las tarjetas de vacunación de todos los usuarios cuyo NSS acabe en ‘0’ de la tabla hash.
- Añadir las tarjetas de vacunación anteriormente borradas de aquellos usuarios cuyo NSS acabe en ‘30’ a la tabla hash.
- Mostrar el número de colisiones máximo que se han producido al volver a insertar las tarjetas de vacunación.
- Vacunar a todos los usuarios cuyo NSS sea par de la primera dosis y mostrar el número de vacunados. Para ello, se debe obtener su tarjeta de vacunación.
- Vacunar de la segunda dosis a todos los usuarios que tengan entre 20 y 50 años.
- Vacunar a todos los usuarios cuyo NSS acabe en 2 o 6 que aún no tengan la pauta completa.
- Comprobar cuántos usuarios tienen pasaporte covid.

De igual forma que en la práctica anterior, si en algún momento del proceso de vacunación un centro se queda sin dosis en almacén, leer del fichero las primeras 100 dosis sobrantes (a partir de la última dosis leída del fichero), añadirlas al multimap y proseguir con la vacunación. Así mismo, mientras haya dosis de las recomendadas se ponen éstas, si no se ponen de cualquier otro fabricante. **Siempre se da prioridad a las dosis que ya están en el CentroVacunación**, por lo que si en un centro no quedan dosis de un tipo determinado, se vacunará con una dosis de otro tipo, independientemente de si hay del tipo recomendado en las dosis no leídas del fichero.

Nota: Para los que trabajan en parejas:

- void THashTarjetaVacunacion::redispersar(unsigned tam), que redispersa la tabla a un nuevo tamaño *tam.*
- Obtener el número de usuarios que tienen una dosis no recomendada y mostrarlo por pantalla.

Nota: Opcional

- Medir la diferencia de tiempos entre buscar en la tabla hash las tarjetas de vacunación frente a realizarlo en un *std::map* de tarjetas de vacunación. Para ello, (1) crear un vector con todas las claves de las tarjetas (crearlas mediante concatenación); (2) buscar todas estas claves en la tabla hash y medir el tiempo total de hacer todas estas búsquedas (3) comentar el uso de la tabla hash en el proyecto y sustituirlo por un mapa para que haga la misma función que la tabla hash (4) volver a buscar todas las claves del vector en el mapa y medir los tiempos. (5) Indicar en un documento estas medidas de tiempo e indicar la diferencia. Hacer todo esto en una función dentro de GestiónVacunas.

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.