Práctica 5. Tabla hash

Sesiones de prácticas: 2

Objetivos

Implementación y optimización de tablas de dispersión cerrada.

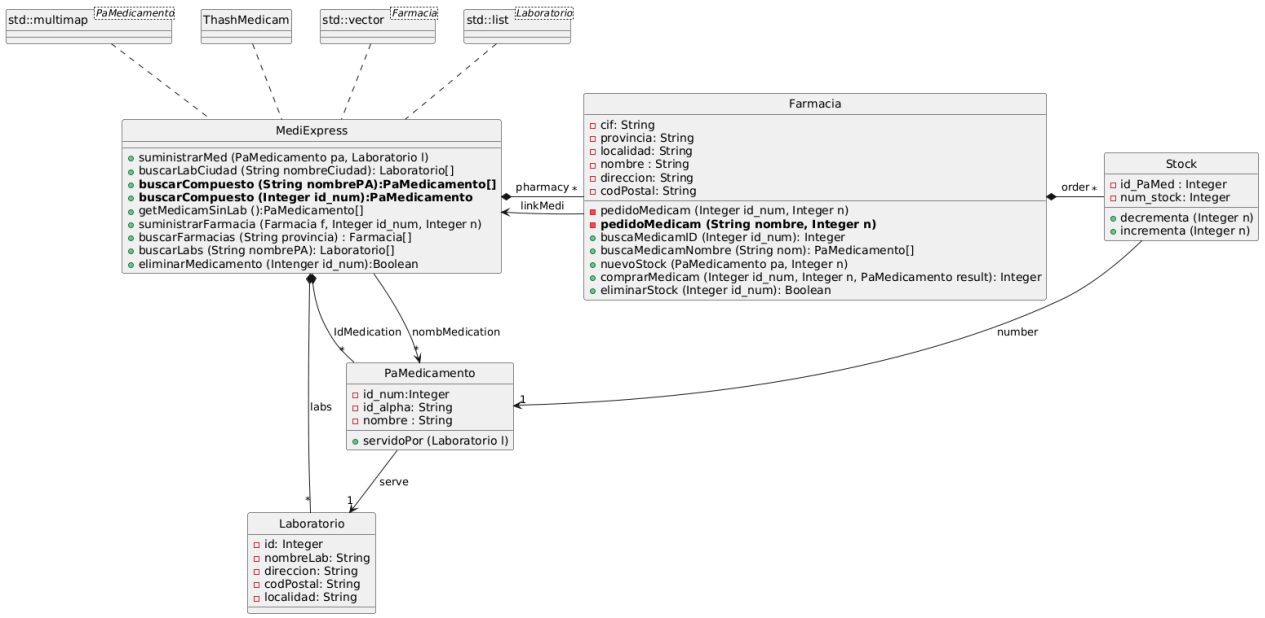
Descripción de la EEDD

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

- *ThashMedicam::hash(unsigned long clave, int intento)*, función privada con la función de dispersión.
- *ThashMedicam::ThashMedicam(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.
- *ThashMedicam::ThashMedicam(ThashMedicam &thash)*. Constructor copia.
- *ThashMedicam::operator=(ThashMedicam &thash)*. Operador de asignación.
- *~ThashMedicam()*. Destructor.
- *bool ThashMedicam::insertar(unsigned long clave, PaMedicamento &pa)*, que inserte un nuevo PaMedicamento 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.
- *Medicam* ThashMedicam::buscar(unsigned long clave)*, que busque un PaMedicamento a partir de su clave de dispersión numérica y devuelva un puntero al PaMedicamento localizado (o *nullptr* si no se encuentra).
- *bool ThashMedicam::borrar(unsigned long clave)*, que borre el PaMedicamento de la tabla. Si el elemento no se encuentra, devolverá el valor falso.
- *unsigned int ThashMedicam::numElementos()*, que devuelva de forma eficiente el número de elementos que contiene la tabla. Guardar dicho valor internamente en 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.

Se actualizará el diseño según el siguiente esquema UML.



Descripción de la práctica

En esta práctica se mantiene la funcionalidad anterior, pero vamos a optimizar las búsquedas por medicamento, ya que no siempre se conoce el ID de la clase PaMedicamento. La modificación principal está en los cambios de contenedores para relacionar las clases. La relación entre MediExpress y PaMedicamento ahora es doble, la composición se mantendrá por ID único con PaMedicamento mediante la tabla hash y la clase ThashMedicam a través de la relación *MediExpress::idMedication*. Esta relación intentará realizar la búsqueda por medicamentos mediante hashing a partir del ID numérico del medicamento con *MediExpress::buscarCompuesto(Integer)*. Sin embargo, para mejorar la búsqueda por cualquier palabra del nombre del medicamento, la relación *MediExpress::nombMedication* utilizará un *std::multimap* en la función *MediExpress::buscarCompuesto(String)*. De este modo, el **MAGNESIO CLORURO HEXAHIDRATO** se descompone en tres palabras. Como hay 21 ocurrencias de *MAGNESIO* en nuestro sistema, el *MAGNESIO* tendrá 21 entradas en total en el multimapa, el *HEXAHIDRATO* tendrá 5 y algunas más el *CLORURO*. En este caso, para buscar la palabra completa, se harán las 3 búsquedas por separado en el multimapa y se guardarán en un *std::set,* de manera que el resultado final sea la intersección de los 3 conjuntos (calcular las intersecciones por pares de conjuntos). En este caso particular, el resultado será PaMedicamento con ID=10904.

La clase Farmacia también añade la función de pedir medicamento por su nombre en con *Farmacia::pedidoMedicam (String)*. Esta función llama a *MediExpress::buscarCompuesto (String)* y hace el pedido de 10 unidades de todos los medicamentos que se obtengan en la búsqueda.

También se cambiarán algunos de los contenedores de la Práctica 4 para que ahora sí, sean realmente eficientes. Los contenedores a usar para las seis relaciones "uno a muchos" son:

- *Farmacia::order* → **std::map<int, Stock>** relación de composición que mantiene el stock y, por tanto, los medicamentos ordenados por ID.
- *MediExpress::pharmacy* → **std::multimap<string, Farmacia>** con clave la provincia.
- *MediExpress::idMedication* →**ThashMedicam** con clave el ID del PaMedicamento

- *MediExpress::nombMedication* → **std::multimap<string, PaMedicamento*>** relación de asociación usando cada palabra del nombre del medicamento como clave.
- *MediExpress::labs* → **std::list<Laboratorio>** para mantener los laboratorios sin orden específico.

La nueva funcionalidad de las clases es similar a la anterior, aunque teniendo en cuenta las características de los nuevos contenedores que deben realizar las búsquedas de forma eficiente siempre que sea posible. El proceso de compra también ahora es algo diferente, cuando un cliente va a la farmacia, primero pregunta por el medicamento (ahora las funciones de búsquedas de medicamento en MediExpress son públicas) y ya puede decidir si espera a que se lo suministren a la farmacia (con *Farmacia::pedidoMedicam()*) o busque otro alternativo.

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 MediExpress::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 MediExpress::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 MediExpress::promedioColisiones()*, que devuelve el promedio de colisiones por operación de inserción realizada sobre la tabla.
- *float MediExpress::factorCarga()*, que devuelve el factor de carga de la tabla de dispersión.
- *unsigned int MediExpress::tamTabla()*, que devuelve el tamaño de la tabla de dispersión.
- *void MediExpress::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 *MediExrpess* 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 *MediExpress* para precargar los datos de los ficheros y, posteriormente, mostrar el estado interno de la tabla con el método *Mediexpress::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).

[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

Pasos a seguir:

1. Implementar la clase *THashMedicam*.
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 λ al tamaño de los datos.
4. Mostrar el estado interno de la tabla con el método *Mediexpress::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 medicamentos utilizando la nueva implementación de la tabla hash. Para ello, durante la lectura del fichero de medicamentos se guardarán sus IDs en un std::vector y luego se buscarán todos esos medicamentos en la tabla hash con *Mediexpress::buscarCompuesto(Integer id_num)*.

Pasos a seguir:

1. Implementar la prueba de rendimiento de la tabla hash.
2. Al mismo tiempo que se lee el fichero de datos, crear una std::list<PaMedicamento> para comparar los resultados con la búsqueda de todos los medicamentos usando dicha lista y documentar la diferencia de tiempos con la tabla hash añadiendo esta información a analisis_THash.md (ver Apartado 2.2 y 2.4).

Programa de prueba 2

A continuación se describen los pasos que debe seguir el **constructor de la clase MediExpress**, que tomará los nombres de los 3 ficheros como parámetros:

1. Leer los ficheros de las prácticas anteriores y cargar la información en los contenedores de la STL *MediExpress:pharmacy* y *MediExpress::labs* y en la nueva tabla hash en *MediExpress::idMedication* con la mejor configuración encontrada en la fase anterior. La búsqueda *buscarCompuesto (Integer id_num):PaMedicamento* se realiza ahora con la tabla hash. Guardar las claves de los medicamentos en un vector auxiliar, *vMedi,* durante el proceso de lectura del fichero de principios activos para el siguiente paso.
2. Para establecer la relación *MediExpress::nombMedication*, buscar todos los identificadores de los medicamentos (están en el vector *vMedi* del paso anterior) en la tabla hash obteniendo el objeto PaMedicamento (su dirección de memoria). Tomar nota del tiempo que se tarda en hacer todas las búsquedas. Luego extraer las palabras que forman el nombre del medicamento por separado e insertar en el multimapa los pair <string, PaMedicamento*> para cada una de estas palabras. La búsqueda *buscarCompuesto (String nombrePA):PaMedicamento[]* se puede hacer ahora con el nombre completo del medicamento.
3. Al igual que en prácticas anteriores, enlazar de forma automatizada los medicamentos (un total de 3310) con los laboratorios (un total de 1579), incluyendo a los 152 ubicados en Madrid. Para ello de nuevo usar el vector *vMedi*, buscar las claves, obtener los PaMedicamento* y hacer el enlace. Enlazar también las farmacias con los medicamentos de forma consecutiva cada 100 medicamentos, como se hizo anteriormente, haciendo que haya inicialmente en stock 10 medicamentos de cada tipo.

4. Para comprobar la eficiencia de la tabla hash con respecto a un *std::list*, introducir todos los PaMedicamentos en una lista auxiliar de prueba (no forma parte del UML) y comprobar los tiempos de hacer la búsqueda de todas las claves del vector *vMedi* en dicha lista.

Una vez que la estructura está inicializada realizar las siguientes acciones:

1. Buscar los siguientes compuestos por nombre completo y mostrarlos en pantalla. El orden de entrada del nombre del medicamento ahora no importa. El resultado de la búsqueda pueden ser más de un medicamento:
    a. MAGNESIO CLORURO HEXAHIDRATO
    b. CLORURO
    c. ANHIDRO CALCIO CLORURO
    d. LIDOCAINA HIDROCLORURO
    e. MENTA PIPERITA
    f. VIRUS GRIPE
2. De nuevo los sevillanos van a comprar magnesio, en concreto 12 personas van a comprar a todas las farmacias de Sevilla algún tipo de "MAGNESIO", da igual cuál es su tipo y comprarán cualquier de ellos disponible en stock. Si al buscar no hay ningún tipo de magnesio en stock, entonces pedirán al laboratorio "ÓXIDO DE MAGNESIO" con ID=3640. Mostrar por pantalla el estado anterior y posterior de cada farmacia de Sevilla tras este proceso.
3. Tras una alerta sanitaria en Úbeda, las autoridades del hospital de la comarca han recomendado a la población en riesgo tomar un antígeno para la mejora del sistema inmune. Mostrar el stock de la única farmacia de Jaén (que está en Úbeda) antes y después de pedir "ANTIGENO OLIGOSACARIDO" en todas sus formas. Para ello, buscar el stock inicial de este antígeno (hay varios) y en cualquier caso pedir 10 unidades de todos los que haya disponibles.
4. En todo el país se ha prohibido que se dispensen CIANURO y el BISMUTO en las farmacias, así que eliminar dichos medicamentos y también el stock que tengan de éstos todas las farmacias. Para ello, buscar primero por nombre, y luego eliminar los medicamentos en el orden correcto.

Nota: Para los que trabajan en parejas:

- Implementar *void MediExpress::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 $\boldsymbol{\lambda}$ determinado lance el método redispersar a un tamaño de la tabla un 30% más grande. Modificar también el método *MediExpress::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.