Universidad de Jaén
Departamento de Informática

Práctica 5. Tabla hash

Sesiones de prácticas: 2

Objetivos

Implementación y optimización de tablas de dispersión cerrada.

Descripción de la EEDD

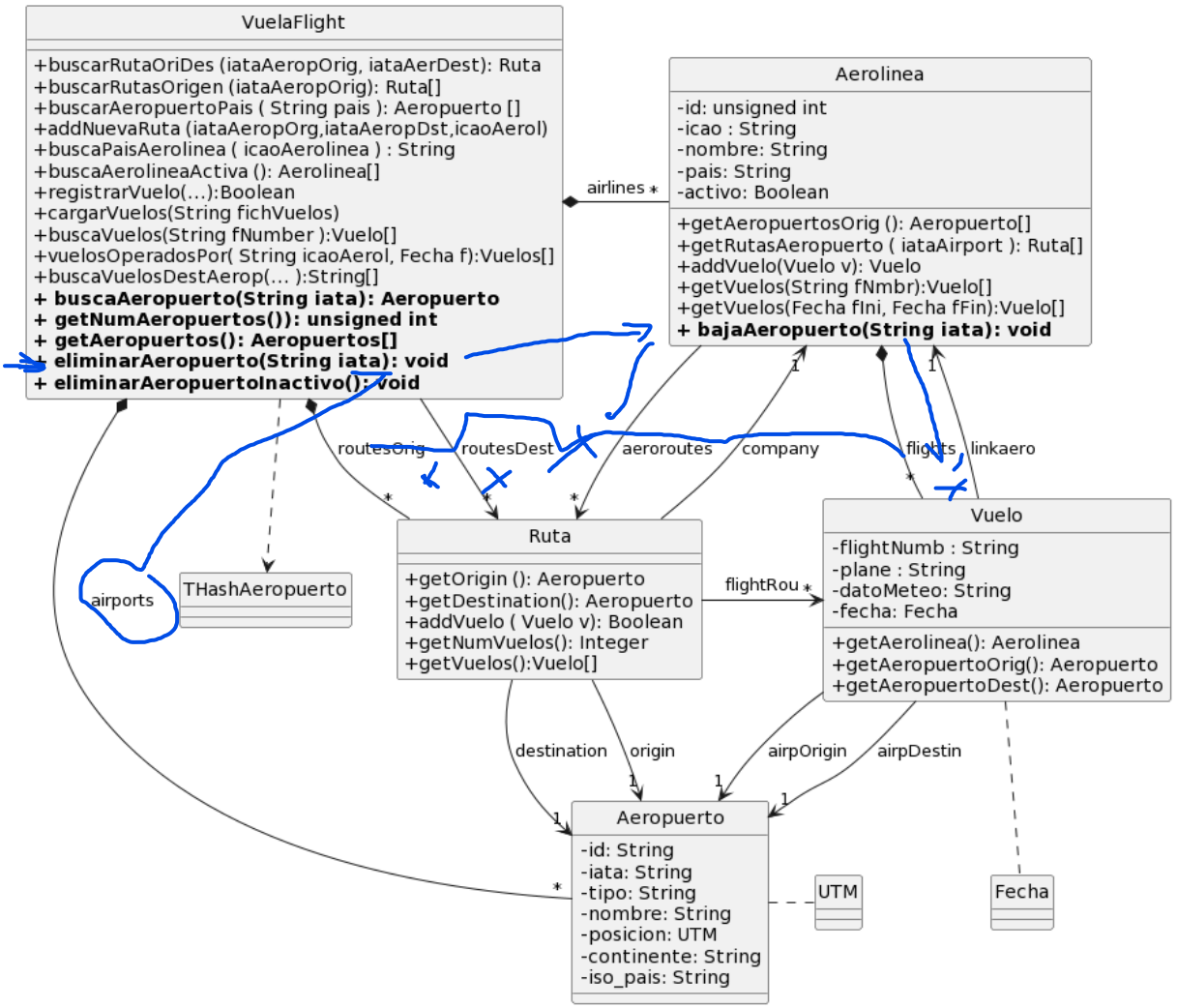
En esta práctica se implementará una tabla hash de dispersión cerrada de aeropuertos, por lo que **no se implementará esta vez mediante un template**. Su definición sigue esta especificación:

- *ThashAerop::hash(unsigned long clave, int intento)*, función privada con la función de dispersión.
- *ThashAerop::ThashAerop(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. maxelementos / lambda tenemos el tamaño a meter en la thash
- *ThashAerop::ThashAerop(ThashAerop &thash)*. Constructor copia.
- *ThashAerop::operator=(ThashAerop &thash)*. Operador de asignación.
- *~ThashAerop()*. Destructor.
- *bool ThashAerop::insertar(unsigned long clave, Aeropuerto &aeropuerto)*, que inserte un nuevo aeropuerto 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.
- *Aeropuerto* ThashAerop::buscar(unsigned long clave)*, que busque un aeropuerto a partir de su clave de dispersión numérica y devuelva un puntero al aeropuerto localizado (o *nullptr* si no se encuentra).
- *bool ThashAerop::borrar(unsigned long clave)*, que borre el aeropuerto de la tabla. Si el elemento no se encuentra devolverá el valor falso. pasar de ocupado a disponible
- *unsigned int ThashAerop::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 *VuelaFlight* mientras se optimizan los contenedores existentes. La implementación de estas mejoras tiene como objetivo principal aumentar la eficiencia en la gestión de datos en *VuelaFlight*.

Una de las modificaciones será el cambio de la relación existente entre *VuelaFlight* y *Aeropuerto* (*VuelaFlight::airports*), transformando el vector de aeropuertos (*std::vector<Aeropuerto>*) en un una tabla hash (***ThashAerop***) para mejorar la eficiencia del programa.

Además, se realizarán ajustes en la relación entre la clase *VuelaFlight* y la clase *Ruta*. Se establecerán dos relaciones, ***VuelaFlight::routesOrig*** y ***VuelaFlight::routesDes***, para almacenar las rutas según el código IATA del aeropuerto de origen o de destino. Esta modificación permitirá realizar búsquedas eficientes buscando tanto por aeropuerto de salida como de llegada, al estar ambos aeropuertos indexados. El contenedor usado para ambas relaciones será un *std::multimap*, utilizando como clave el IATA del aeropuerto. En el caso de la DATO clave, se hará uso de tipos específicos (***Ruta*** y ***Ruta****) como valores para las rutas de origen y destino, respectivamente, en los *std::multimap*.

Resumiendo, los contenedores a usar para las siete relaciones son los siguientes:

- Vuelaflight::airports→ *ThashAerop,* para optimizar el acceso a los aeropuertos.

- Vuelaflight::routesOrig→ *std::multimap<std:string, Ruta>*, almacena rutas según el código IATA del aeropuerto de origen.
- Vuelaflight::routesDes→ *std::multimap<std:string, Ruta*>*, almacena rutas según el código IATA del aeropuerto de destino.
- Vuelaflight::airlines→ *std::map<string, Aerolinea>* con clave ICAO de la aerolínea
- Aerolinea::flights→ *std::multimap<string,Vuelo>* con clave el identificador de vuelo
- Aerolinea::aeroroutes→*std::deque<Ruta*>*
- Ruta::flightRoute→ *std::list<string, Vuelo*>*

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

Aerolínea

- *bajaAeropuerto(IATAAirport): void*, dado el iata de un aeropuerto elimina de la aerolínea toda su información relativa a rutas comerciales relacionadas y vuelos vinculados a dicho aeropuerto. Utilizar desde VuelaFlight::*eliminarAeropuerto()*

VuelaFlight

- *buscaAeropuerto(IATAAirport): Aeropuerto*, devuelve un aeropuerto a partir de su IATA
- *getNumAeropuertos(): unsigned int*, devuelve el número de aeropuertos registrados
- *getAeropuertos(): Aeropuertos[], devuelve todos los aeropuertos registrados en el sistema*
- *eliminarAeropuerto(std::string IATA): void,* dado un código IATA de un aeropuerto, elimina dicho aeropuerto y su información relacionada en el sistema: rutas existentes y vuelos. *Nota: utilizar método Aerolinea::bajaAeropuerto para eliminar también las referencias a rutas comerciales y vuelos de aquellas aerolíneas que lo hayan utilizado*
- *eliminarAeropuertoInactivo(): void,* elimina todos los aeropuertos inactivos, comprobando que realmente están inactivos mediante la comprobación de la existencia o no existencia de rutas para ese aeropuertos. Si existiera alguna ruta asociada a algún aeropuerto inactivo (poco probable) eliminar su información relacionada en el sistema (ruta y vuelos existentes).

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 ThashAerop::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 ThashAerop::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 ThashAerop::promedioColisiones()*, que devuelve el promedio de colisiones por operación de inserción realizada sobre la tabla.
- *float ThashAerop::factorCarga()*, que devuelve el factor de carga de la tabla de dispersión.
- *unsigned int ThashAerop::tamTabla()*, que devuelve el tamaño de la tabla de dispersión.
- *void VuelaFlight::mostrarEstadoTabla()* que muestra por pantalla los diferentes parámetros anteriores de la tabla interna de aeropuertos. Usar el método en main después de llamar al constructor de *VuelaFlight* 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**, una vez sustituido el vector de aeropuertos por la tabla en *VuelaFlight*, y realizados los cambios en la tabla para cada experimento, en main utilizar únicamente el constructor de *VuelaFlight* para precargar los datos de los ficheros y, posteriormente, mostrar el estado interno de la tabla con el método *VuelaFlight::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.

Pasos a seguir:

1. Implementar la clase *THashAerop* de acuerdo al diagrama UML y realizar todas las modificaciones relativas a las relaciones entre clases indicadas en el enunciado.
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 VuelaFlight::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 aeropuertos utilizando la nueva implementación de la tabla hash. Para ello, se obtendrán todos los aeropuertos en un vector y se obtendrán $N=10^6$ números aleatorios dentro del rango de posiciones del vector. Los números aleatorios obtenidos representan la posición dentro del vector de aeropuertos, y con cada uno de esos códigos seleccionados, se buscará su clave correspondiente en la tabla hash, usando el método *VuelaFlight::buscaAeropuerto()*

Pasos a seguir:

1. Implementar la prueba de rendimiento de la tabla hash.
2. Obtener los aeropuertos de VuelaFlight y obtener los números aleatorios.

---

[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

3. Realizar las búsquedas correspondientes de aeropuertos en VuelaFlight (almacenados en la tabla hash ThashAerop) y medir los tiempos.
4. Comparar los resultados con la búsqueda de los mismos aeropuertos mediante un mapa y documentar la diferencia de tiempos. Para ello, será necesario instanciar un std::map en main con los mismos aeropuertos con los que hemos instanciado ThashAerop, obtenidos con VuelaFlight::getAeropuertos().
5. Documentar y analizar los resultados obtenidos durante la prueba añadiéndolos al final del informe analisis_THash.md.

Programa de prueba 2

Crear un programa de prueba con las siguientes indicaciones:

1. Instanciar *VuelaFlight* 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 el aeropuerto con IATA "00AS" y mostrar sus datos. Tras esto, eliminar el aeropuerto con dicho código. Volver a buscar el aeropuerto borrado y, si no se encuentra, volver a insertarlo en el sistema. Mostrar el estado de la tabla para comprobar el número de colisiones máximo que se han producido al volver a insertar el aeropuerto.
4. Eliminar todos los aeropuertos inactivos con el método implementado eliminarAeropuertoInactivo(). Comprobar el número de aeropuertos registrados antes y después de realizar esta operación además del estado de la tabla.

Nota: Para los que trabajan en parejas:

- Implementar *void ThashAerop::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 *VuelaFlight::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.