Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2019-2020

Práctica 4: Implementación de búsqueda mediante Tabla Hash

1. Objetivo

En esta práctica se trabaja los algoritmos de búsqueda interna, tanto la implementación en lenguaje C++ como el estudio de la complejidad computacional de dichos algoritmos.

2. Entrega

Esta práctica se realizará en dos sesiones en las siguientes fechas:

> Sesión tutorada: 31 de marzo de 2020, online a las 8:30 y a las 14:40;
> Sesión entrega: presencial(*) en su grupo los días 13, 14,15 y 16 de abril de 2020.
> (*)Si continúa el estado de alerta será online el 16 de abril a las 8:30 y a las 14:30.

Durante las sesiones se podrán proponer modificaciones y mejoras en el enunciado de la práctica.

3. Enunciado

Desarrollar un tipo de dato genérico en lenguaje C++ que implemente la técnica de búsqueda basada en tabla hash con dispersión cerrada [1].

Realizar un estudio empírico del rendimiento de la tabla hash cuando se utilizan distintas funciones de dispersión, distintas estrategias de exploración y distintos factores de carga. En este estudio se incluirán las funciones de dispersión: módulo, suma y pseudoaleatoria; y para las estrategias de exploración: lineal, cuadrática, dispersión doble y re-dispersión.

El estudio del rendimiento de la tabla hash requiere desarrollar un programa en C++ que cuente el número de operaciones de comparación de clave que se realizan durante una operación de búsqueda en la tabla hash. El programa utilizará un contador de comparaciones, que se inicializa a cero antes de cada búsqueda en la tabla hash y se incrementa con cada ejecución de una operación de comparación de claves. Al finalizar cada búsqueda en la tabla hash el valor contenido en el contador se utiliza para actualizar una estadística que registra los valores mínimo, máximo y media del número de comparaciones obtenidos. Para que estos valores estadísticos sean significativos el experimento de búsqueda de una clave en la tabla hash debe repetirse un número suficiente de veces. En la sección de notas de implementación se indica el formato de visualización de los datos obtenidos en la ejecución del programa.

El estudio del rendimiento de la tabla hash consiste en realizar distintas ejecuciones del programa desarrollado con una tabla con capacidad para 100, 1.000 y 10.000 claves variando el factor de carga entre los valores (0.3, 0.5, 0.7, 0.8, 0.9) y anotar los valores mínimo, media y máximo del número de comparaciones obtenidos en cada ejecución.

De forma similar se pueden estudiar las variaciones del comportamiento de la tabla hash al modificar otros parámetros: número de celdas, número de claves por celda, función de dispersión o función de exploración de la tabla hash. Para estos estudios se mantiene fijo el valor del factor de carga.

4. Notas de implementación

Desarrollar en lenguaje C++ la plantilla de clases `Tabla<Clave>` que implemente las siguientes operaciones:

- `Buscar(Clave X)`: retorna el valor booleano `true` si el valor `X` del tipo `Clave` está guardado en la tabla hash. En otro caso retorna `false`.
- `Insertar(Clave X)`: retorna el valor booleano `true` si se añade el valor `X` del tipo `Clave` a la tabla hash. En otro caso retorna `false`.

Y los siguientes atributos:

- Vector de celdas (`vCelda`): este atributo de la clase contiene las celdas de la tabla hash. El número de celdas de la tabla (`nCeldas`) se especifica mediante un parámetro del constructor de la tabla. Este valor coincide con el número de posibles valores retornados por una función de dispersión.
- Función de dispersión (`fDispersion`): este atributo de la clase implementa la función de dispersión que utiliza la tabla hash. Recibe como parámetro un valor `X` del tipo `Clave` y retorna un valor en el intervalo `[0..nCeldas-1]` que indica la posición de la celda dentro de `vCelda` que debería contener valor `X` del tipo `Clave`. El constructor de la tabla hash recibe un parámetro que le indica la función de dispersión a instanciar.
- Función de exploración (`fExploracion`): este atributo de la clase implementa la estrategia de exploración que utiliza la tabla hash. Recibe como parámetros un valor `X` del tipo `Clave` y el número del intento de exploración `(i>0)`, que se incrementa en cada llamada a esta función. En el i-ésimo intento de exploración la función retorna el desplazamiento, respecto a la posición dada por la función de dispersión, de la celda dentro de `vCelda` que debería contener valor `X` del tipo `Clave`. El constructor de la tabla hash recibe un parámetro que le indica la función de exploración a instanciar.

***

Las celdas de la tabla hash se implementan mediante la plantilla de clases `Celda<Clave>`. Cada celda almacena un número fijo (`nClaves`) de valores del tipo `Clave`, que se especifica como un parámetro en los constructores de las clases `Tabla<Clave>` y `Celda<Clave>`. La clase genérica `Celda<Clave>` implementa las siguientes operaciones:

- `Buscar(Clave X)`: retorna el valor booleano `true` si el valor `X` del tipo `Clave` está guardado en la celda. En otro caso retorna `false`.
- `Insertar(Clave X)`: retorna el valor booleano `true` si se añade el valor `X` del tipo `Clave` a la celda. En otro caso retorna `false`.
- `estaLlena()`: retorna el valor booleano `true` si los `nClaves` posiciones de la Celda están ocupadas. En otro caso retorna `false`.

***

En la implementación de las funciones de dispersión y exploración se utilizará polimorfismo dinámico para poder elegir en tiempo de ejecución, al crear la tabla hash, las funciones a usar. Por tanto, los atributos `fDispersion` y `fExploración` de la clase `Tabla<Clave>` serán punteros a una clase base abstracta a partir de la cual se derivan las clases que implementan cada método. En este caso, el método nulo de la clase base será el operador de llamada a función:

```
virtual int FDispersionBase<Clave>::operator()(const Clave&)=0;
virtual int FExploracionBase<Clave>::operator()(const Clave&, int i)=0;
```

***

Para realizar el estudio del comportamiento de la tabla hash se utilizarán valores de clave del tipo `DNI`. Un valor del tipo `DNI` está formado por la concatenación de ocho dígitos numéricos, por tanto, los posibles valores están en el rango `[00000000...99999999]`. La clase `DNI` implementa los siguientes métodos:

- `DNI()`: constructor por defecto, que inicializa el objeto DNI con un valor generado aleatoriamente.
- `operator==()`: para utilizar la clase `DNI` como `Clave` se deben sobrecargar todos los operadores de comparación utilizados en las plantillas `Tabla<Clave>` y `Celda<Clave>`.
- `operator unsigned long()`: para la conversión a valor numérico un valor de la clase `DNI` se recomienda sobrecargar el operador de conversión al tipo unsigned long.

***

El programa principal realizará la siguiente secuencia de pasos:

1. Solicita los parámetros para crear una tabla hash:
    a. Número de celdas, `nCeldas`. El número de posiciones de la tabla hash.
    b. Tamaño de la celda, `nClaves`. El número de claves que se pueden almacenar en cada celda.
    c. Función de dispersión, `fDispersion`. Opciones: módulo, suma y pseudoaleatoria.
    d. Función de exploración, `fExploracion`. Opciones: lineal, cuadrática, dispersión doble y re-dispersión
2. Solicita los parámetros del experimento:
    a. Factor de carga, `factor`. Valor entre 0 y 1 que se corresponde al cociente entre el número de valores de clave almacenados y el número de valores que es posible almacenar en la tabla.
    b. Número de pruebas, `nPruebas`. Número de repeticiones de la operación, inserción o búsqueda, que se realiza en el experimento.
3. Crear un banco de prueba con `2*N` valores de tipo `DNI` generados de forma aleatoria. El banco de pruebas se guarda en un vector, con tamaño `N=factor*nCeldas*nClaves`.

4. Insertar en la tabla hash los `N` primeros valores sacados del banco de prueba hasta alcanzar el factor de carga indicado.

5. El experimento para estudiar el comportamiento de la operación de búsqueda consiste en:

    a. Inicializar a cero los contadores de comparaciones de claves. Valores mínimo, acumulado y máximo.

    b. Realizar la búsqueda de las `nPruebas` claves extraídas de forma aleatoria de las primeras `N` claves del banco de prueba, o sea, de las claves que están guardadas en la tabla hash. Para cada búsqueda se cuenta el número de comparaciones realizadas, y se actualizan los valores mínimo, máximo y acumulado.

    c. Al finalizar el experimento se presentan los valores mínimo, máximo y medio del número de comparaciones de claves contabilizados.

6. El experimento para estudiar el comportamiento de la operación de inserción se basa en contar el número de comparaciones para buscar claves que no se encuentran en la tabla. Consiste en:

    a. Inicializar a cero los contadores de comparaciones de claves. Valores mínimo, acumulado y máximo.

    b. Realizar la búsqueda de las `nPruebas` claves extraídas de forma aleatoria de las `N` últimas claves del banco de prueba, las que no están guardadas en la tabla hash. Para cada búsqueda se cuenta el número de comparaciones realizadas, y se actualizan los valores mínimo, máximo y acumulado.

    c. Al finalizar el experimento se presentan los valores mínimo, máximo y medio del número de comparaciones de claves contabilizados.

A continuación se muestra el formato de salida con los resultados de la ejecución:

```
Celdas nClaves    Dispersión Exploración     Carga      Pruebas
xxxx   xxxx       xxxxxxx    xxxxxx          xxxx       xxxx

           Mínimo      Medio      Máximo
Búsquedas   xxxx       xxxx       xxxx
Inserción   xxxx       xxxx       xxxx
```

5. Referencias

[1] Apuntes de clase.