Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2021-2022

Práctica 4: Búsqueda por dispersión

1. Objetivo

El objetivo de esta práctica es trabajar con los algoritmos de búsqueda vistos en las clases de teoría [1] y realizar una implementación en lenguaje C++ de las técnicas de dispersión. Se utilizará la definición de tipos genéricos, el polimorfismo dinámico y la sobrecarga de operadores.

2. Entrega

Se realizará en dos sesiones de laboratorio en las siguientes fechas:

Sesión tutorada: del 28 de marzo al 1 de abril de 2022
Sesión de entrega: del 4 al 8 de abril de 2022

3. Enunciado

Se pide desarrollar un tipo de dato genérico `HashTable<class Key>` para contener valores de tipo clave, `Key`, utilizando la técnica de búsqueda por dispersión de claves. La implementación del tipo genérico debe permitir especificar:

- El tamaño de la tabla de dispersión, `tableSize`. Es el número de posiciones en la tabla.
- Una función de dispersión. Mapea el valor de la clave *k* a una posición de la tabla. Se pide implementar, al menos, las siguientes funciones de dispersión:
    - Módulo, *h(k) = k % tableSize*
    - Basada en la suma, *h(k) = Sum(k_i) % tableSize*
    - Pseudoaleatoria, *h(k) = {srand(k); rand()}*.
- La forma de resolver las colisiones en la tabla de dispersión [2]. Se pide implementar las técnicas de:
    - Dispersión abierta, cada posición de la tabla hash contiene una lista donde se insertan los ***sinónimos***, registros de claves distintas a las que les corresponde esa posición en la tabla según la función de dispersión utilizada.
    - Dispersión cerrada, cada posición de la tabla hash contiene un array que establece el número máximo de registros que pueden almacenarse en dicha posición.
- En el caso de dispersión cerrada [1]:
    - Tamaño del bloque, `blockSize`. Este valor representa el máximo número de registros que se pueden almacenar en la misma celda de la tabla.
    - Estrategia de exploración para resolver el desbordamiento en un bloque. Se pide implementar, al menos, las siguientes funciones de exploración (donde *k* es el valor de la clave, y el parámetro *i* es el número de intentos de exploración).
        - Exploración lineal, *g(k,i) = i*
        - Exploración cuadrática, *g(k,i) = i^2*
        - Doble dispersión, *g(k,i) = f(k) * i*
        - Redispersión, *g(k,i) = $f^{(i)}(k)$*.

      Las funciones *f(k)* y *$f^{(i)}(k)$* son funciones de transformación adecuadas.

4. Notas de implementación

1. Para implementar los distintos comportamientos de la función de dispersión se define una familia de clases genéricas con una clase base abstracta, `DispersionFunction<Key>` que encapsula la sobrecarga del operador función como un método nulo.

```
template<class Key>
class DispersionFunction {
  public:
    unsigned operator()(const Key& k) const = 0;
};
```

   De esta forma, la implementación de cada función de dispersión particular se realiza en la sobrecarga del operador función en una clase derivada. Por ejemplo, la función de dispersión *Módulo*:

```
template<class Key>
class fdModule: public DispersionFunction<Key> {
  public:
    fdModule(const unsigned n): tableSize(n){}
    unsigned operator()(const Key& k) const {
      return k % tableSize;
    }
  private:
    unsigned tableSize;
};
```

2. Para implementar las distintas estrategias de exploración se utiliza la misma estrategia que con la función de dispersión. Se encapsula la definición de la función de exploración dentro de la clase genérica abstracta `FuncionExploracion<Key>` que sobrecarga al operador función como un método nulo.

```
template<class Key>
class ExplorationFunction {
  public:
    unsigned operator()(const Key& k, unsigned i) const = 0;
};
```

   De esta forma, la implementación de cada función de exploración particular se realiza en la sobrecarga del operador función de una clase derivada. Por ejemplo, la función de exploración lineal:

```
template<class Key>
class feLineal: public FuncionExploracion<Key> {
  public:
    unsigned operator()(const Key& k, unsigned i) const {
      return i;
    }
 };
```

En la implementación de la función de exploración cuadrática sólo se tendrán en cuenta los desplazamientos positivos respecto a la posición inicial. En la función de exploración de dispersión doble, la función de dispersión *f(k)* utilizada para la exploración se pasará como parámetro en el constructor de la clase derivada. Para la implementación de la estrategia de exploración de redispersión se utilizará el generador de números pseudo-aleatorios. Se inicializa la semilla del generador con el valor de la clave, y se utiliza el valor de la *i*-ésima llamada a `rand()` como el valor del desplazamiento $f^{(i)}(k)$ respecto a la posición inicial.

3. Cada posición de la tabla hash contiene un objeto de la clase genérica abstracta `Sequence<Key>` donde se almacenan los valores de clave sinónimos. En esta clase genérica abstracta se definen los siguientes métodos nulos:

    a. Método público `bool search(const Key& k) const`, que retorna el valor booleano `true` si el valor `k` del tipo `Key` está guardado en la secuencia. En otro caso retorna el valor `false`.

    b. Método público `bool insert(const Key& k)`, retorna el valor booleano `true` si se inserta el valor `k` del tipo `Key` en la secuencia. En otro caso, si el valor ya estaba guardado en la secuencia, se retorna el valor booleano `false`.

    c. Método público `bool isFull() const`, retorna el valor booleano `true` si la secuencia está llena. En otro caso retorna el valor booleano `false`.

4. Para implementar la técnica de dispersión abierta, donde todos los valores de clave sinónimos se almacenan en la misma posición de la tabla hash, se implementa la clase genérica `List<Key>` que deriva de `Sequence<Key>`.

5. Para implementar la técnica de dispersión cerrada, donde el número de claves sinónimas en cada posición de la tabla hash está acotado por el tamaño del bloque, `blockSize`, se implementa la clase genérica `Block<Key>` que deriva de `Sequence<Key>`.

6. En la clase genérica `HashTable<Key>` se definen los siguientes miembros:

    a. Atributo privado `tableSize`, contiene el tamaño de la tabla. Se especifica en el constructor.

    b. Atributo privado `table`, es un array de `tableSize` posiciones en cada una de las cuales hay un puntero a un objeto `Sequence<Key>` que almacena las claves sinónimas.

    c. Atributo privado `fd`, es un puntero a la clase base `DispersionFunction<Key>` que apunta al objeto que implementa la función de dispersión a utilizar. Se especifica en el constructor de la tabla.

    d. Atributo privado `blockSize`. En la técnica de dispersión cerrada contiene el tamaño del bloque utilizado, mientras que en la técnica de dispersión abierta se le asigna el valor `0`, que será su valor por defecto en el parámetro del constructor de la tabla.

    e. Atributo privado `fe`, es un puntero a la clase base `ExplorationFunction<Key>` que apunta al objeto que implementa la función de exploración a utilizar en la técnica de dispersión cerrada. Mientras que en la técnica de dispersión abierta se le asigna el

valor `nullptr`, que será su valor por defecto en el parámetro del constructor de la tabla.

f. Método público `bool search(const key& k) const`, retorna el valor booleano `true` si el valor `k` del tipo `Key` está guardado en la tabla hash. En otro caso retorna el valor booleano `false`.

g. Método público `bool insert(const Key& k)`, retorna el valor booleano `true` si puede insertar el valor `k` del tipo `Key` en la tabla hash. En otro caso, si el valor ya está guardado en la tabla, se retorna el valor booleano `false`.

h. Método `Constructor` recibe como parámetros:

- El tamaño de la tabla que se utiliza para inicializar el atributo `tableSize`.
- Un puntero a un objeto del tipo función de dispersión, que se enlaza al atributo `fd`.
- Un puntero a un objeto del tipo función de exploración, que se enlaza al atributo fe. Este parámetro tiene el valor por defecto `nullptr`.
  - Cuando se invoca al constructor con el valor por defecto en este parámetro, la tabla hash debe funcionar con la técnica de dispersión abierta. Para ello, crea en memoria dinámica un array de objetos de la clase `List<Key>` que asigna al atributo `table`.
  - Cuando se invoca al constructor con un valor del parámetro distinto de `nullptr`, la tabla hash debe funcionar con la técnica de dispersión cerrada. Para ello, crea en memoria dinámica un array de objetos de la clase `Block<Key>` que se asigna al atributo `table`.
- El tamaño del bloque, que tiene el valor de defecto `0`. Se utiliza para inicializar el atributo `blockSize` cuando la tabla hash funciona con la técnica de dispersión cerrada. Cuando la tabla funciona con la técnica de dispersión abierta este atributo debe tener su valor por defecto.

7. El programa principal tiene el siguiente comportamiento:

   a. Se pide al usuario el tamaño de la tabla, la función de dispersión y la técnica de dispersión a utilizar.

   b. En caso de elegir la técnica de dispersión cerrada también se pide al usuario el tamaño del bloque y la función de exploración a utilizar.

   c. Se crea el objeto que implementa la función de dispersión elegida, y si es el caso también se crea el objeto que implementa la función de exploración.

   d. Se crea una tabla hash con el tipo de `Clave=long` y los parámetros especificados por el usuario.

   e. A continuación implementa un menú que permite insertar y buscar los valores de clave indicados por el usuario, y muestra el resultado de la operación.

5. Referencias

[1] Google: Apuntes de clase.
[2] Wikipedia: Tabla hash: https://es.wikipedia.org/wiki/Tabla_hash