Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2024-2025

Práctica 4: Búsqueda por dispersión

1. Objetivo

El objetivo de esta práctica es trabajar los algoritmos de búsqueda vistos en la teoría [1], realizar una implementación de las técnicas de dispersión y comprobar de forma empírica la complejidad computacional de las técnicas estudiadas. La implementación en lenguaje C++ utiliza la definición de tipos genéricos (plantillas), el polimorfismo dinámico y la sobrecarga de operadores.

2. Entrega

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

Sesión tutorada: del 17 al 20 de marzo
Sesión de entrega: del 24 al 27 de marzo

3. Enunciado

La técnica de búsqueda por dispersión [1] se implementa mediante la estructura de datos conocida como tabla de dispersión, o **tabla hash** [2]. Esta implementación de la tabla hash debe tener opciones de configuración para funcionar con las siguientes variantes de la técnica:

- Por simplicidad, asumimos que el registro de información almacenado en la tabla de dispersión coincide con el tipo de la clave utilizada `Key`.
- El tamaño de la tabla de dispersión, que en adelante denotamos `tableSize`, indica el número de posiciones en la tabla. Como hemos visto, este parámetro no sólo indica la cantidad de memoria que se reserva al construir la tabla, sino que también afecta al comportamiento de la función de dispersión y la función de exploración.
- La función de dispersión mapea el valor de la clave k de tipo `Key` a una posición de la tabla en el rango `[0..tableSize-1]`. 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()}*.
- Para resolver las colisiones en la tabla de dispersión, se pide implementar las técnicas:
  - Dispersión abierta, cada posición de la tabla hash contiene una estructura de datos dinámica, tipo lista, donde se insertan todos los ***sinónimos***, generados por la función de dispersión.
  - Dispersión cerrada, cada posición de la tabla hash contiene una estructura de datos estática, tipo array, donde se establece el número máximo de registros que pueden almacenarse en dicha posición.
- En la implementación de la técnica de dispersión cerrada se definen los siguientes parámetros adicionales:
  - El tamaño del bloque, que denotamos `blockSize`, indica el máximo número de registros que se pueden almacenar en la misma posición de la tabla.

- La estrategia de exploración para resolver el ***desbordamiento*** en un bloque debe implementar, al menos, las siguientes funciones de exploración, donde `k` es el valor de la clave, y el parámetro `i` es el número del intento 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 dispersión adecuadas para generar un valor de desplazamiento en el rango `[1..tableSize-1]`.

4. Notas de implementación

Para desarrollar un tipo de dato genérico que permita configurar el funcionamiento de las técnicas de búsqueda por dispersión se implementan los siguientes tipos de datos auxiliares.

1. La elección de la función de dispersión requiere definir la siguiente familia de clases genéricas.

   a. En la clase base abstracta, `DispersionFunction<Key>`, se define la sobrecarga del operador función como un método nulo que recibe un parámetro del tipo `Key` y retorna una posición de la tabla.

   ```
   virtual unsigned DispersionFunction<Key>::operator()(const Key&) const =
                                                                          0;
   ```

   b. Cada clase derivada implementa una de las funciones de dispersión solicitadas. Los parámetros adicionales requeridos por la función de dispersión, como `tableSize`, se pasan por parámetro en el constructor de la clase derivada.

2. De forma similar, se define una familia de clases genéricas que implementan las distintas estrategias de exploración.

   a. En la clase base abstracta, `ExplorationFunction<Key>`, se define la sobrecarga del operador función como un método nulo con dos parámetros, un valor del tipo `Key` y el número del intento de exploración. El método retorna el desplazamiento hasta la siguiente posición de la tabla a explorar.

   ```
   virtual unsigned ExplorationFunction<Key>::operator()(const Key&,
                                                         unsigned) const=0;
   ```

   b. Cada clase derivada implementa una de las estrategias de exploración solicitadas. Los parámetros adicionales requeridos en la función de exploración se pasan por parámetro en el constructor de la clase derivada.

   c. En la implementación de la función de exploración cuadrática sólo se tienen en cuenta los desplazamientos positivos respecto a la posición inicial.

   d. En la implementación de la función de exploración por dispersión doble, la función de dispersión auxiliar $f(k)$ se pasa por parámetro al constructor de la clase derivada.

   e. En la implementación de la estrategia de exploración por redispersión se utiliza el generador de números pseudo-aleatorios. Se inicializa la semilla del generador con

el valor de la clave, `srand(k)`, y se retorna el valor de la *i*-ésima llamada a `rand()` como el valor de desplazamiento $f^{(i)}(k)$.

3. Para permitir elegir entre las técnicas de dispersión abierta y dispersión cerrada, en cada posición de la tabla hash se almacena un puntero a la clase genérica abstracta `Sequence<Key>`. En esta clase base se definen las siguientes operaciones:

    a. Método que busca el valor de clave `k` del tipo `Key` en la secuencia. Si lo encuentra retorna el valor booleano `true`; en otro caso retorna el valor `false`.

    ```
    virtual bool Sequence<Key>::search(const Key& k) const = 0;
    ```

    b. Método que inserta el valor de clave `k` del tipo `Key` en la secuencia. Si lo puede insertar retorna el valor booleano `true`; en otro caso retorna el valor `false`.

    ```
    virtual bool Sequence<Key>::insert(const Key& k) = 0;
    ```

4. A partir de la clase base `Sequence<Key>` se derivan los tipos de secuencia específicos, cada uno especializado en implementar una de las técnicas de dispersión: abierta y cerrada.

    a. En la técnica de dispersión abierta, donde todos los valores de clave sinónimos que generan colisión se almacenan en la misma posición de la tabla hash, se requiere una implementación de la secuencia basada en una estructura de datos dinámica.

    ```
    template<class Key> class dynamicSequence: public Sequence<Key>;
    ```

    b. En la técnica de dispersión cerrada, la implementación de la secuencia utiliza una estructura de datos estática. El número máximo de valores de clave que pueden almacenarse en cada posición de la tabla, `blockSize`, se pasa como parámetro al constructor de la secuencia.

    ```
    template<class Key> class staticSequence: public Sequence<Key>;
    ```

    c. En una secuencia con un número máximo de valores de clave se requiere un método que retorna el valor booleano `true` si la secuencia está llena; en otro caso retorna el valor booleano `false`.

    ```
    virtual bool staticSequence<Key>::isFull() const;
    ```

5. La tabla de dispersión se implementa mediante un tipo genérico que almacena registros del tipo clave `Key` utilizando una tabla donde cada posición contiene un objeto del tipo `Container`. Cualquier tipo de dato usado como parámetro `Container` debe implementar la definición del tipo abstracto `Sequence<Key>`.

    ```
    HashTable<class Key, class Container=staticSequence<Key> >
    ```

6. Usaremos el tipo `staticSequence<Key>` como valor por defecto para el tipo de contenedor. Esto indica que el comportamiento por defecto de la tabla de dispersión se corresponde con la técnica de dispersión cerrada.

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

    a. Atributo privado `tableSize` que contiene el tamaño de la tabla.

b. Atributo privado `table`, es el array con `tableSize` posiciones. Cada posición contiene un puntero a un objeto del tipo `Container`.

c. Atributo privado `fd`, es una referencia a la clase `DispersionFunction<Key>`. Se inicializa con el objeto del tipo función de dispersión elegida.

d. Atributo privado `fe`, es una referencia a la clase `ExplorationFunction<Key>`. Se inicializa con el objeto del tipo función de exploración elegida.

e. Atributo privado `blockSize` que contiene el tamaño del bloque. Se utiliza para construir los objetos del tipo `Container`.

8. La clase genérica `HashTable<Key,Container>` implementa los métodos definidos en la clase genérica abstracta `Sequence<Key>`. Por tanto, se puede utilizar como otra implementación particular de secuencia.

9. El método constructor recibe los parámetros necesarios para inicializar sus atributos.

```
HashTable<Key,Container>::HashTable(unsigned, DispersionFunction<Key>&,
                                    ExplorationFunction<Key>&,unsigned);
```

a. El tamaño de la tabla que se utiliza para inicializar el atributo `tableSize`.

b. Un objeto del tipo función de dispersión para inicializar el atributo `fd`.

c. Un objeto del tipo función de exploración para inicializar el atributo fe.

d. El tamaño del bloque que se utiliza para inicializar el atributo `blockSize`.

e. El atributo `table` se inicializa creando un objeto `Container` en memoria dinámica para cada una de sus posiciones del array de tamaño `tableSize`.

10. Para implementar la técnica de dispersión abierta se utilizará una especialización parcial de la plantilla cuando se utiliza el tipo `dynamicSequence<Key>` como parámetro `Container`.

```
template<class Key> class HashTable<Key, dynamicSequence<Key> >
```

11. Para la implementación de la técnica de dispersión abierta no se requieren los atributos función de exploración y tamaño del bloque, utilizados por las estrategias de exploración. Por tanto, el constructor de la clase sólo requiere los parámetros:

```
HashTable<Key,dynamicSequence<Key> >::HashTable(unsigned,
                                    DispersionFunction<Key>&);
```

a. El atributo `table` es un array con `tableSize` posiciones, cada una de las cuales contiene un objeto `dynamicSequence<Key>`.

12. El programa principal aceptará las siguientes opciones por línea de comandos:

a. `-ts <s>`, `s` es el tamaño de la tabla.

b. `-fd <f>`, `f` es el código que identifica a una función de dispersión.

c. `-hash <open|close>`, indica la técnica de dispersión a utilizar. Implica el tipo de secuencia utilizada para el parámetro `Container`.

d. `-bs <s>`, `s` es el tamaño del bloque. Sólo para dispersión cerrada.

e. `-fe <f>`, `f` es el código que identifica a una función de exploración. Sólo para dispersión cerrada.

13. 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.

14. Se crea la tabla hash, con la versión adecuada de la plantilla según el valor indicado en la opción `-hash`, invocando al constructor con los parámetros indicados por línea de comandos. Se utiliza una clase `nif` como tipo de dato `Key` en la plantilla.

15. El tipo de dato `nif` representa la parte numérica del número de identificación fiscal. Encapsula un número entero de 8 dígitos, e implementa las operaciones necesarias para poder utilizarlo como tipo de dato Key.

    a. El constructor por defecto crea e inicializa un dato `nif` con un valor aleatorio.

    b. Constructor de cambio de tipo a partir de un dato entero `long`.

    c. Sobrecarga de los operadores de comparación utilizados.

    d. Sobrecarga del operador de cambio de tipo, que permite convertir un dato de tipo `nif` en un entero para realizar las operaciones numéricas.

```
nif::operator long();
```

16. 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