UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ciencias

Estructuras de Datos

Práctica 15: Hash

Profesora:
Amparo López Gaona
Ayudante: Adrián Aguilera Moreno
Ayudante de Laboratorio: Kevin Jair Torres Valencia

Objetivos

- Implementen una tabla hash genérica que almacene pares clave-valor.
- Diseñen y comparen dos funciones de dispersión distintas:
  - Función basada en el método de división (residuo)
  - Función basada en corrimiento de bits
- Desarrollen estrategias para el manejo de colisiones.
- Implementen iteradores para permitir el recorrido eficiente de los elementos almacenados.

Introducción

La estructura de datos de la tabla de dispersión (tabla hash) es una matriz de cubetas o *buckets* que almacena los pares clave/valor en ellos. La función hash ayuda a determinar la ubicación de una clave determinada en la lista de depósitos.

Generalmente, el código hash es un número entero no negativo que es igual para objetos iguales y puede o no ser igual para objetos desiguales. Es posible que dos objetos desiguales tengan el mismo código hash. A esto se le llama colisión. Para resolver colisiones, generalmente se utiliza una matriz de listas. Los pares asignados a un solo bucket (índice de matriz) se almacenan en una lista y la referencia de la lista se almacena en el índice de matriz.

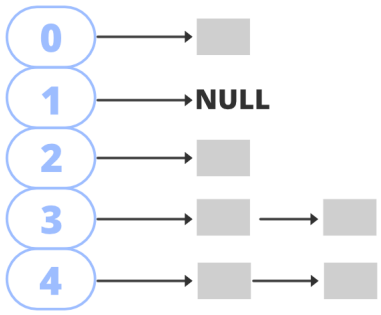

Figure 1: Representación de mapeo de claves a índices de una tabla

Las tablas de dispersión (tablas hash) representan una de las estructuras de datos más eficientes para implementar diccionarios, permitiendo operaciones de inserción, búsqueda y eliminación en tiempo constante promedio ($O(1)$). Generalmente, su funcionamiento se basa en tres componentes fundamentales:

- Función de dispersión (hash): Transforma la clave en un índice dentro del arreglo interno.
- Arreglo de buckets (cubetas): Almacena los elementos en posiciones determinadas por la función hash.
- Mecanismo de manejo de colisiones: Resuelve situaciones donde múltiples claves se asignan al mismo índice.

Dentro de Java, existe un método `hashCode()` que nos ayuda a determinar qué cubeta (bucket) debe mapear el par clave/valor.

Funciones de dispersión

La función de dispersión (hash) toma una clave k y devuelve un índice $h(k)$ que corresponde a una posición válida dentro de de la tabla de tamaño m.

$$h(k) = posicion\ en\ la\ tabla$$

Existen muchas funciones/técnicas de dispersión:

- Dispersión por módulo (hashing modular):

  $$h(k) = k\ mod\ n$$

  Se utiliza cuando la clave es un número entero. Ejemplo: Si $k = 123$ y $m = 10$, entonces

  $$h(k) = 123\ \ mod\ 10 = 3$$

  Como desventaja Puede tener patrones si m no es un número primo (todos o la mayoría de los datos terminan con el mismo dígito).

- Dispersión por plegamiento (Folding): Divide la clave en dos o más partes (por ejemplo, dígitos o bytes), las suma o combina de alguna manera, y luego aplica el módulo. Ejemplo: Para la clave 123456, se divide en partes: $12 + 34 + 56 = 102$ y luego se hace $102\ \ \text{mod}\ 2$.

  Como ventaja es muy útil para claves largas, además permite transformar claves no numéricas a valores enteros y como desventaja si no se combina bien puede perder información.

- Dispersión por multiplicación (Cadenas de caracteres): Multiplicar cada caracter de una cadena por una potencia de 37 y sumarlo. Expresado formalmente como:

  $$\sum_{i=0}^{n} elemento[i]37$$

con n = tamaño de la cadena

- Función con corrimientos: Toma cada carácter de una cadena de caracteres y acumula su valor multiplicándolo por 2, sólo que no utiliza la multiplicación, si no corrimientos. Al final divide esta suma entre el tamaño de la tabla y se devuelve el residuo de esta división.

Manejo de colisiones

Cuando dos claves distintas tienen el mismo valor de hash $h(k)$, ocurre una colisión. Las técnicas para manejar colisiones incluyen:

- Encadenamiento: Cada posición de la tabla guarda una lista enlazada con todos los elementos que colisionan.
- Dirección abierta: Si una posición está ocupada, se busca otra libre.

Desarrollo

En esta práctica, deberán completar la implementación de un Árbol AVL genérico, siguiendo el principio de separación de interfaz e implementación. Se te proporcionarán algunos archivos base que deberás extender.

Para ello tendrán que implementar los métodos en las siguientes clases:

- `HashImpl`: Como estructura interna se tiene:
  - Arreglo de listas enlazadas: Cada celda de la tabla (`LinkedList<ParClaveValor<K, V»`) almacena pares clave-valor.
  - Redimensionamiento: Cuando el factor de carga (elementos/capacidad) supera 0.75, se duplica el tamaño del arreglo.

  operaciones:

  - `agregar(clave, valor)`: Calcula el índice con la función de dispersión. Si la clave ya existe, actualiza el valor. De otro caso, añade el nuevo par a la lista.
  - `obtener(clave)`: Busca en la lista correspondiente al índice calculado.
  - `eliminar(clave)`: Remueve el par de la lista si existe.

  Métodos adicionales:

  - `codigoHash()`: Suma los códigos hash de todos los elementos (para verificación de integridad).
  - `existeValor(valor)`: Recorre toda la tabla buscando coincidencias.

- Funciones de Disperción:
  - `FuncionDivision:` Usa el operador % (modulo) para distribuir las claves. Ejemplo: hash("Pikachu") = $3326234\%10 = 4$ (para capacidad = 10).
  - `FuncionCorrimiento:` Mezcla bits altos/bajos para evitar colisiones en claves similares. Ejemplo: hash("Pikachu") = $(3326234 \wedge (3326234 >>> 16))\%10 = 2$.
- `IteradorHash<K, V>`: Iterador que recorre los pares clave-valor en la tabla.

Archivos proporcionados

- Interfaces
  - `HashInterfaz<K, V>`: Interfaz con las operaciones básicas (`agregar, obtener, eliminar`, etc.).
  - `FuncionDispersion`: Define el comportamiento de las funciones hash
  - `Iterador`: Patrón para recorrido de elementos.
- Implementaciones:
  - `HashImpl`: Lógica central de la tabla hash.
  - `FuncionDivision/FuncionCorrimiento`: FuncionDivision/FuncionCorrimiento: Estrategias de dispersión.
  - `IteradorHash<K, V>`: Iterador que recorre los pares clave-valor en la tabla.
- Clase de prueba (Main)

Formato de Entrega

1. Las prácticas se entregarán en parejas.
2. NO incluir los archivos .class dentro de la carpeta.
3. Los archivos de código fuente deben estar documentados.
4. Se pueden discutir y resolver dudas entre los integrantes del grupo. Pero cualquier práctica plagiada total o parcialmente será penalizada con cero para los involucrados.
5. La práctica se debe subir al Github Classroom correspondiente.
6. La entrega en classroom debe contener el link HTTPS y SSH de su repositorio y es lo único que se debe entregar.
7. El horario y día de entrega se acordará en la clase de laboratorio y no deberá sobrepasar 2 clases de laboratorio.