EDA GII - ETSInf - Curso 2018-2019

Práctica 3 - Parte 2: Uso de una Tabla de Dispersión como implementación eficiente de un Map

Departamento de Sistemas Informaticos y Computación
Universitat Politècnica de València

1. Objetivos formativos y trabajo previo

El objetivo de esta segunda parte de la práctica es la implementación eficiente de una *Tabla Hash*. En concreto, se completará la implementación de una Tabla Hash con encadenamiento separado (Hashing enlazado) con:

- las operaciones básicas para calcular su factor de carga, obtener una lista de sus claves y hacer *rehashing* cuando la eficiencia de la tabla se vea comprometida.
- las operaciones para comprobar la efectividad de las funciones de `hashCode` definidas sobre las claves.

Recuérdese que dos son los factores que intervienen en el coste de las operaciones `insertar`, `recuperar` y `eliminar` de una Tabla de Dispersión: el coste del cálculo del `indiceHash(c)` asociado a la clave, y el coste de la búsqueda dinámica en la *ListaConPI* `elArray[indiceHash(c)]`. Según ya se ha estudiado en las clases de teoría, las propiedades de la Función de Dispersión son las siguientes:

- Efectividad: debe producir una dispersión efectiva de las claves en las distintas listas.
- Eficiencia: debe de poder ser calculada de forma eficiente, ya que las operaciones básicas sobre la tabla requieren su cálculo.
- No es inyectiva: se pueden producir colisiones.

Para que se puedan implementar las operaciones en tiempo constante el cálculo de `indiceHash` debe realizarse, a su vez, en tiempo constante y, para ello, las longitudes de las listas deben estar acotadas por una constante, por ejemplo un valor menor que 2. Para estudiar la efectividad de la función de dispersión utilizada se calculará la desviación típica de las cubetas y se analizará el histograma de ocupación de las mismas que describe la forma en la que la función consigue repartir los elementos entre las cubetas.

Además se reforzarán los objetivos transversales a todas las prácticas de la asignatura y que están relacionados con la calidad de los programas desarrollados: utilización de paquetes para facilitar la organización, reutilización y mantenimiento del software, utilización de los mecanismos de herencia y genericidad que proporciona el lenguaje, elaboración de juegos de prueba para validar código y generación de documentación asociada al código desarrollado. Para aprovechar al máximo la sesión de laboratorio, antes se debe realizar una lectura comprensiva de este boletín y del código de las clases que se proporcionan a través de PoliformaT.

2. Implementación de una Tabla de Dispersión: la clase `TablaHash`

En las clases de teoría se ha estudiado la *Tabla de Dispersión* como una implementación eficiente de un *Map*. En concreto, se ha diseñado la clase `EntradaHash`, que representa un par clave-valor en la tabla y parcialmente la clase `TablaHash` que implementa el hashing enlazadado mediante un array en el que cada cubeta se representa mediante una `ListaConPI`. En este apartado se propone que el alumno complete esta clase con métodos para obtener una `ListaConPI` con sus claves y hacer *rehashing*. También debe añadir métodos que permitan razonar sobre la eficiencia de la Tabla Hash, en particular sobre el número de cubetas y sobre la efectividad de la función de *hashCode* definida sobre sus claves.

Actividad #4: ubicación de las clases `EntradaHash` y `TablaHash`

Ubica en *librerias.estructurasDeDatos.deDispersion* las clases `TablaHash`, `EntradaHash` y las de prueba `TestTablaHash1` y `TestTablaHash2`, disponibles en *PoliformaT*.

Actividad #5: los métodos `factorCarga()`, `claves()` y `rehashing()`

Completa la clase `TablaHash` implementando los siguientes métodos:

- `double factorCarga()`: devuelve el factor de carga de la tabla.
- `ListaConPI<C> claves()`: devuelve una *ListaConPI* que contiene las claves del *Map* actual.
- `void rehashing()`: incrementa la capacidad del array para mantener el rendimiento de la *Tabla Hash*. La nueva capacidad del array deberá ser igual al siguiente número primo del doble de la capacidad actual.

Una vez resueltos los errores de compilación y revisados los errores de estilo (opción *checkstyle* de BlueJ), comprueba que el código desarrollado es correcto utilizando la clase `TestTablaHash1`.

Actividad #6: los métodos `desviacionTipica()` e `histograma()`

Implementa en la clase `TablaHash` los siguientes métodos:

- `double desviacionTipica()`: devuelve la desviación típica de las longitudes de las cubetas. Si denotamos por l_i la longitud de la cubeta i, la desviación típica σ se define como sigue, asumiendo que hay N cubetas y que la longitud media de las cubetas es $\bar{l}$:

$$\sigma = \sqrt{\frac{\sum (l_i - \bar{l})^2}{N}}$$

- `String histograma()`: devuelve un String que representa el histograma de ocupación de las cubetas de la tabla, donde cada línea consta de dos valores separados por un tabulador: longitud de cubeta y número de cubetas de esa longitud. En dicho histograma, las longitudes de cuberta a considerar son 0, 1, ... 8 y mayores o iguales que 9.

Una vez resueltos los errores de compilación y revisados los errores de estilo (opción *checkstyle* de BlueJ), comprueba que el código desarrollado es correcto utilizando la clase `TestTablaHash2`.

3. Estudio de la efectividad de la función de dispersión

En este apartado se estudiarán las consecuencias de una elección adecuada de la función de dispersión; para ello, se propone:

1. Implementar funciones de dispersión alternativas sobre cadenas de caracteres, envolviendo la clase `String` en la propia `Termino` y sobrescribiendo en ella los métodos `equals` y `hashCode()` de `Object`.
2. Diseñar e implementar código Java para evaluar el comportamiento de las diferentes funciones de dispersión (efectividad en la dispersión y eficiencia).
3. Mostrar gráficamente el histograma de ocupación de la tabla en cada caso y razonar sobre la efectividad de la dispersión en base a la información obtenida sobre éste y la desviación típica, en concreto observando el número de cubetas que tienen uno o dos elementos.

3.1. Implementación de funciones de dispersión sobre cadenas

La clase `Termino` se ha definido como una envolvente de la clase `String`. Un objeto de tipo `Termino` tiene un dato que es un `String` y su valor de hashing asociado que vendrá dado por la función de dispersión que se haya seleccionado al construir el objeto. Se trata de implementar los métodos privados para la obtención del valor de Hash, `hashCode()`, de la clase `Termino`. La implementación debe permitir utilizar diferentes funciones de dispersión (funciones que generan un valor numérico a partir de un `String`). Se deben considerar al menos cuatro funciones de dispersión, la primera sólo suma los códigos de los caracteres del `String`, sin pesos; el resto incorporan diferentes pesos: la propuesta en el libro de Weiss usa 37, la que propone Mckenzie 4 y la del estándar de java usa 31:

Simple:	$s_0 + s_1 + \ldots + s_{n-1}$
Weiss:	$s_0 * 37^{n-1} + s_1 * 37^{n-2} + \ldots + s_{n-1}$
McKenzie:	$s_0 * 4^{n-1} + s_1 * 4^{n-2} + \ldots + s_{n-1}$
String:	$s_0 * 31^{n-1} + s_1 * 31^{n-2} + \ldots + s_{n-1}$

Actividad #7: la clase `Termino`

Descarga desde poliformaT la clase incompleta `Termino` en el paquete `biblioteca` y completa los siguientes:

1. Métodos privados de cálculo del valor de hashing según las diferentes funciones de dispersión descritas: *Simple*, *Weiss*, *McKenzie* y *String*. Estas funciones se deben implementar de forma eficiente, evitando calcular potencias (i.e. sin usar ni `Math.pow` ni `Math.exp`).
2. Método público `hashCode()` que sobrescribe el de `Object`;
3. Método público `equals(Object o)` que sobrescribe el de `Object`.

A continuación usa el CodePad de BlueJ para crear objetos de tipo `Termino` con las diferentes funciones de dispersión y comprobar que los valores de hashing son los correctos. En la siguiente tabla puedes ver algunos ejemplos:

palabra	Simple	Weiss	McKenzie	String
saco	422	5961662	9419	3522362
asco	422	5074550	8555	3003422
noreste	768	-1911759776	602277	2127397360
enteros	768	700909144	564879	-1591951684
cronista	867	621309751	2239905	2118401189
cortinas	867	1733334423	2232087	-452686651

Actividad #8: la clase `EvaluaFuncionDispersion`

Para comprobar la efectividad de las cuatro funciones de dispersión definidas se deberá:

1. Añadir en el paquete `biblioteca` la clase incompleta `EvaluaFuncionDispersion`.
2. Completarla para comparar la calidad de las funciones de dispersión diseñadas utilizando como datos los términos (o palabras) que se encuentran en los documentos indicados en el fichero `lista10.txt`. El método privado `construirTabla(String listaLibros, int fdis)` lee los ficheros de texto que se encuentran en `listaLibros`, extrae los términos y los guarda en una `TablaHash<Termino, Termino>` utilizando la función de dispersión que se indica, (`fdis`). Los ficheros de documentos se encuentran ubicados en `asigDSIC/libros/TXT`. En concreto, el `main` de la clase debe hacer:

   a) Construir la Tabla Hash a partir de los términos encontrados en los libros de la lista `lista10.txt` usando la funcion de hashing correspondiente.

   b) Mostrar por pantalla los valores de factor de carga y desviación típica de cada tabla.

   c) Generar el histograma de cada tabla y guardarlo en el directorio `res` de la carpeta `aplicacionesbiblioteca` con el nombre del método de hashing utilizado precedido de `histo` y acabado en `.txt`, por ejemplo `histoSimple.txt`, `histoWeiss.txt`, etc.

3. Utilizando la herramienta `gnuplot`, dibujar los histogramas obtenidos para cada función de dispersión:

   ```
   gnuplot> plot "histogXXXX.txt" using 1:2 with boxes
   ```

   donde `histoXXXX.txt` hace referencia al fichero que resulta cuando se utiliza la función de dispersión `XXXX`.

   A la vista de los resultados obtenidos se razonará sobre cuál es la función de dispersión más adecuada.

Actividad #9: uso de la clase `TablaHash` desarrollada en la aplicación `GUIBiblioteca`

Ya para terminar, comprueba que, utilizando ahora como tipo dinámico el `Map` el código que has desarrollado para la clase `TablaHash`, la aplicación `GUIBiblioteca` sigue funcionando correctamente.