Implementación de una tabla de Hash

Elaborado por: Arnaldo Quintero Segura.

Documentación técnica

Para la implementación de esta Tabla de Hash, se procederá a desarrollar cada una de las operaciones que se puede realizar utilizándola. Y una ligera explicación de por qué es de esta manera. Esta tabla, dadas las especificaciones de la actividad, soporta claves de tipo `String` unicamente, mientras que los valores pueden ser de cualquier tipo genérico `T` que el usuario desee. Para permitir, si el usuario desea, que todos los elementos sean de un mismo tipo; en caso de querer utilizar tipos variados, se puede utilizar `Object` como valor genérico.

Antes de entrar en detalle en la tabla de hash, veamos la estructura ayudante que se utiliza. Así como el valor constante de números primos.

Pair<K,T>

Esta clase, es una estructura de ayuda, que contiene dos atributos: clave `key`, y valor `value`. De manera que representa un par ordenado. Dónde la clave puede tener un tipo genérico `K` y el valor `T`. Para permitir de esta manera mayor versatilidad a momento de uso.

Ambos atributos poseen sus respectivos `getter` y `setter` para que el usuario pueda interactuar con la clave y valor del par.

De igual manera, el constructor recibe dos parámetros: clave y valor, para una inicialización directa de un par ordenado.

PrimeNumbers

Esta clase, únicamente contiene un atributo estático con los valores de los números primos entre 2 y 997. Esto como ayuda para el proceso de redispersión que se explicará posteriormente.

Propiedades de la tabla de hash

La tabla de hash, posee únicamente dos atributos: `buckets` y `size`, los cuales representan las entradas de la tabla, y la cantidad de elementos que posee insertada la tabla correspondientemente.

Dado que el usuario no debe manejar las entradas de la tabla, el atributo se mantiene privado para único uso interno. De manera contraria, el valor `size` posee su método `getSize()`, para que el usuario pueda saber cuántos elementos hay actualmente insertados en la tabla.

De igual manera, la tabla posee el método `getLoadFactor()`, el cual calcula y retorna el factor de carga actual de la tabla.

Las entradas de la tabla son representadas con listas enlazadas, de tipo `LinkedList`. La cual contiene elementos de tipo `Pair<String,T>`. Siendo asi listas enlazadas de pares ordenados de `String` y un genérico `T`.

De esta manera, el atributo `buckets`, es un array de tamaño fijo, que comienza en `3`. Y que a medida que se va insertando o eliminando, teniendo en cuenta el factor de carga. Realiza una redispersión para aumentar o disminuir la cantidad de entradas que posee la lista.

Insertar

Para insertar un elemento a la tabla de hash, existe el método `put(key, value)`, el cual recibe como parámetros la clave y valor que desean ser insertados. Y devuelve un valor de tipo `boolean`, el cual corresponde a si la respuesta fue exitosa o no.

En caso de que la clave no se encuentre en la tabla, posee a insertarse. En caso contrario, se retorna el valor `false` y la tabla no se ve modificada. En caso de querer modificar el valor, se debe utilizar el método `replace` explicado más adelante.

Como función de hash para esta tabla, se utiliza el método `String.hashCode()`, de la clase `String` de `Java`. Para de esta manera obtener un entero a partir de un string. Luego, se procede a calcular el valor absoluto del módulo en tamaño `size`. Para que sea una posición válida dado el número de entradas. Este proceso es realizado por la función `getIndex(key)`, el cual recibe una clave de tipo `String` y retorna la posición en la cual debe ser insertada.

Esta función posee una variante, para la cual se pasa el tamaño `size`, asumiendo que se está realizando la redispersión. Y el atributo `size` contiene la cantidad vieja de entradas.

Al momento de inserción, se procede a chequear el valor del factor de carga. Y en caso de que sea mayor a `1`, se procede a hacer una redispersión. Para esto, se procede a buscar el número primo más cercano a el doble de el tamaño de entradas que posee la tabla. Para esto se utiliza la clase `PrimeNumbers`, para no tener que calcular los números primos. En caso de que el número buscado sea mayor a `997`, se utiliza directamente el doble de la cantidad de entradas de la tabla.

Eliminar

Para eliminar un elemento de la tabla, se debe conocer previamente la clave a eliminar. Se utiliza la función `remove(key)`, la cual dada la clave. Consigue el elemento y lo elimina de la entrada en la que se encuentre.

Para buscar el elemento dentro de las entradas, se procede a calcular el valor del indice asociado a dicha clave. Y dentro de la lista ordenada comparar cada uno de los elementos hasta conseguir la clave que fue pedida. Esto es realizado por la función privada `hasKey(key)`.

Luego de la eliminación, se procede a revisar el factor de carga, y en caso de ser menor a `0,33` se procede a realizar una redispersión. En la cual el nuevo tamaño será el siguiente primo más cercano al doble de la cantidad de elementos que posee insertada la tabla.

Este método retorna un valor de tipo `boolean` que indica si la eliminación a sido correcta o no.

Reemplazar

Este método sirve para reemplazar el valor asociado a una clave ya insertada en la lista. En caso de que la clave no se encuentre, la lista no se ve modificada.

Esta operación es realizada por el método `replace(key, newValue)`. La cual procede a buscar el par asociado a dicha clave y cambiar el valor asignado por el nuevo valor dado. Para conseguir el par ordenado, se utiliza la función `hasKey` mencionada anteriormente.

Buscar

Función que dada una clave, retorna el valor asociado a la misma. La función `get(key)`. Utiliza la función `hasKey` para obtener el par asociado a la clave dada. Y retorna el valor de tipo `T` asociado a dicho par. En caso de que la clave no se encuentre en la tabla, el método retorna `null`.

Contiene

Este tipo de operación posee dos variantes, `containsKey` y `containsValue`, los cuales buscan si la lista contiene la clave o valor dado correspondientemente.

En caso de buscar por clave, se utiliza la función `hasKey` y se retorna un valor de tipo `boolean` que expresa si la clave se encuentra o no en la tabla.

Para el otro caso, existe la función `hasValue`, la cual se encarga de iterar por cada una de las entradas de la tabla. Y a su vez por cada uno de los elementos. Y retornar la primera clave que consiga que tenga dicho elemento como valor.

En caso de no conseguir el elemento, la función retorna el valor `null`.

Imprimir

Para la impresión de la tabla, tenemos el método `print()` el cual imprime por la salida estándar cada uno de los valores de la tabla en formato `(clave, valor)`, dentro de llaves (`{ }`), separados por coma.

Para esto se utiliza una sobreescritura del método `toString`, para que de esta manera el elemento en sí posea una representación gráfica. Así como su método para impresión directa.

Prueba de funcionalidad, ejemplo real

Para probar la funcionalidad, se incluye el ejecutable `HashTable.jar`, el cual nos muestra cómo una tabla de hash puede ser utilizada para simular una agenda telefónica. En la cual los nombres son las claves, y los números los valores a guardar.

Dentro del ejemplo se procede a demostrar el uso de cada una de las operaciones explicadas anteriormente.