Estructura de Datos (2023-24, Grupo A)

Tabla Hash mediante Prueba Lineal

Tabla Hash mediante Prueba Lineal

En esta práctica tendrás que implementar una Tabla Hash que gestione las colisiones mediante **Prueba Lineal** (Linear Probing). Este tipo de tablas almacenan los datos directamente en un array, sin usar listas enlazadas para las colisiones. El array se considera de forma circular, de modo que cada celda tiene una celda sucesora, siendo la sucesora de la última celda la primera.

El siguiente [fichero comprimido] contiene parte de la implementación. Para realizar la práctica debes editandar el fichero **LinearProbingHashTable.java**.

La tabla que implementaremos almacena asocioaciones de claves y valores (cada clave almacenada en la tabla tendrá un valor asociado). Para ello, usaremos dos arrays:

```
private K keys[];
private V values[];
```

de modo que el primero almacena las claves de la tabla y el segundo los valores. Estos dos arrays son paralelos, lo cual significa que el valor asociado a la clave que se encuentre en la celda i del array keys estará almacenado en la celda i del array values.

La estructura también alamacena un entero (size) que debe corresponder en todo momento con el número de asociaciones almacenadas en la tabla.

En este tipo de tablas, las colisiones se resuelven del siguiente modo: si la celda donde hay que colocar un nuevo elemento ya está ocupada, se intenta colocar en la sucesora, y este proceso se repite hasta encontrar una celda libre. Este modo de gestionar las colisiones debe tenerse también en cuenta a la hora de buscar elementos y al eliminarlos. Al buscar, no basta con comprobar la celda que corresponda al valor hash, sino que hay que examinar también las sucesoras, hasta que o bien se localize el elemento buscado, o bien se encuentre una celda libre (que significa que el elemento buscado no está en la tabla). Al borrar, hay que mover a la nueva posición que le corresponda cada uno de los elementos situados en el bloque de sucesores contiguos al eliminado, es decir, todo aquellos que van desde el sucesor del eliminado hasta la primera celda libre. Esto hará que el proceso de búsqueda siga funcionando de forma correcta. Se recomienda que consultes las transparencias finales del tema 5, donde se muestran distintos ejemplos sobre el funcionamiento de las operaciones.

Las operaciones sobre la tabla se deben implementar tal como se describen a continuación.

searchIdx

Ésta es una operacion privada de la clase que resulta útil para implementar las demás operaciones de la estructura. Se trata de un método que toma una clave y devuelve:

- el índice de la celda de la tabla donde se encuentra dicha clave, si ésta ya estaba en la tabla, o
- el índice de la celda donde debe colocarse la clave, si ésta no está aún en la tabla.

Para ello, se debe calcular el valor hash de la clave usando el método hash. Sea idx dicho valor. Empezando desde la celda idx, se repite el siguiente proceso:
- comprobar si la celda está libre o si ya contiene la clave a insertar, y en dicho caso devolver dicho índice.
- en caso contrario, pasar a la celda sucesora.

Consideraremos que una celda está libre si su contenido es null, por lo que todas las celdas del array keys deben inicializarse con este valor. Afortunadamente, Java ya realiza esta inicialización de celdas al crear el array.

insert

Antes de insertar un elemento, hay que comprobar si el factor de carga de la tabla excede el máximo permitido y, en dicho caso, habrá que realizar el procedimiento de rehashing. Este procedimiento se encuentra ya implementado en el código que se suministra, pero debes estudiarlo para comprender su funcionamineto.

La operación insert tiene un doble significado:

- Si la clave no se encuentra en la tabla, se debe insertar una nueva asociación con dicha clave y su valor asociado.
- Si la clave ya está en la tabla, se debe sobreescribir el valor asociado almacenado en la tabla con el nuevo valor proporcionado.

Para implementar esta operación deberás hacer uso del método searchIdx descrito previamente.

search

Dada una clave, devuelve el valor asociado que le corresponde en la tabla, o null si la clave no estuviese en la tabla. Esta operación también se debe implementar usando searchIdx.

delete

Dada una clave, debe borrar la asociación que le corresponde de la tabla. Si la clave no estuviese en la tabla, ésta no se modifica. Recuerda que para dejar una celda libre tendrás que asignar null a su celda en el array keys.

Una complicación a la hora de implementar esta operación es la siguiente. Dado que, en caso de colisiones, un elemento no tiene que estar almacenado en la celda que corresponde a su valor hash, sino que puede estar almacenado en alguna de las celdas consecutivas no libres, y de esto depende la correción del proceso de búsqueda, para eliminar una asociación de la tabla habrá que mover a sus nuevas posiciones todas las asociaciones adyacentes (todas aquellas que se encuentren a continuación de la eliminada, hasta encontrar un celda libre). Para ello, se debe realizar el mismo proceso que se hace en la insercción con cada uno de los elementos adyacentes, liberando además las celdas antiguas.

Iteradores

Los distintos iteradores que recorren las claves, los valores o ambos se encuentran casi implementados en el código suministrado. Estudia este código y completa el método advance la clase TableIter para completar la implementación.

Para comprobar la correción de tu implementación, puedes ejecutar la clase suministrada HashTableTest.java, que compara el funcionamiento de tu tabla con el de una tabla con encadenamiento enlazado.

Última modificación: viernes, 16 de diciembre de 2016, 15:37

◄ Códigos

Saltar a...

Códigos para el algoritmo de Dijkstra ►