CS367 Programming Assignment 4

Lecture 1, Spring 2018

Due by 11:59 pm on Tuesday, April 10, 2018

In this page: Overview | Specifications | Handing in **Related pages:** Assignments

Overview

Why are we doing this program?

Description

In this assignment you will be implementing a hashtable that uses chaining for collision handling (the chains will be implemented using `LinkedLists`). The `HashTable` class you write will allow the user to specify the initial table size, the maximum load factor, and the maximum chain length and will resize itself automatically. The hashtable will also be able to generate statistics about its current state and you will use the statistcs generated to give you insight into how differences in the table size, max load factor, and max chain length affect the hashtable's performance.

Goals

The goals of this assignment are to:

- Gain experience with hashing.
- Gain experience resizing a data structure.
- Analyze the results of your code to better understand how your program behaves.

Specifications

What are the program requirements?

The `HashTable` Class

The javadoc for the `HashTable` class gives information about the methods you need to implement. Additionally, your hashtable **must**:

- be implemented using an array,
- handle collisions using chaining, and
- use `LinkedLists` to implement the chains.

There are two constructors for the hashtable. One constructor for the hashtable takes two arguments. The first is the initial size of the hashtable. The second is a value that indicates the maximum *load factor* of the table. The load factor is defined to be the number of items in the hashtable divided by the size of the hashtable (i.e., the length of the array). The maximum load factor is a measure of how full the hashtable is allowed to get before it will be resized. For example, a maximum load factor of 0.8 means that, under ideal hashing (when there are no collisions), the hashtable is allowed to get up to 80% full before it will be resized. The second constructor for the hashtable takes three argument: the initial size of the hashtable, the maximum load factor, and the the maximum chain length. The maximum chain length is the largest any chain in the hashtable is allowed to get before the table will be resized.

The index you will use to determine where to store an entry will be very simple. You will call the `hashCode()` method on the item you need to insert or lookup and modulo this value by the table

size (to make sure it is a valid index). One potential problem is the fact that `hashCode()` can return a negative value. For example, when you take a modulo of a negative value, the result is a negative value. An easy way to deal with this is to check the value of `hashCode()` modulo table size. If it is negative, simply add table size to the result. This will give you a positive value between 0 and *table size-1*.

The `dump` method dumps (i.e., prints) the values in the hashtable to a given output source. It works by going through the indices in the hashtable in order (0, 1, 2, ... up to *table size-1*). At each index it prints the the index followed by the values stored at that index. The values at each index should be printed in the order they are stored in the `LinkedList`, starting at the beginning of the `LinkedList` (i.e., index 0). The sample output shows what is required. All the output will go to the `PrintStream` passed as an argument.

The `displayStats` method will display some statistics about the current state of your hashtable. When it is called it should display: the current table size, the number of items in the table, the current load factor, the length of the largest chain (i.e., the maximum number of items stored at any one index), the number of chains of length 0 (i.e., the number of "empty" indices), and the average length of the chains of length greater than 0. The sample output shows what is required. All the output will go to the `PrintStream` passed as an argument.

Note the following:

- You can insert the same item twice. This makes adding an item faster since you don't have to check if it is already there.
- Only non-`null` items can be added to the hashtable. A `NullPointerException` is thrown if `null` is passed as a parameter to `insert`.
- `lookup` returns an item (of the generic type `T`), not a boolean. That is, if it finds the item, it returns it; if not, it returns `null`. The item may be in the hashtable more than once, however, `lookup` should just return the first one it finds at a given index.
- `delete` returns an item (of the generic type `T`), not a boolean. That is, if it finds the item, it returns it (and removes it); if not, it returns `null`. Note that the item may be in the hashtable more than once. However, `delete` should just remove and return the first one it finds at a given index.
- The constructors of the `HashTable` throw `IllegalArgumentExceptions` if the parameters passed to them are not valid (see `HashTable` constructor documentation).
- Your code can (and should) use methods from the `Object` class, such as `equals`, `hashCode`, and `toString`.
- You don't ever need to shrink your hashtable.

The `LinkedList` javadoc lists (and describes) all the `LinkedList` methods. You can use any of the `LinkedList` methods so be sure to look over the documentation carefully to determine which `LinkedList` methods you will want to use in this program.

Analyzing the results

In the file `Questions.txt` you will find some questions that you will need to answer. These questions are meant to have you think about the results you get from the code you are writing.

Testing

The driver class `TestHash` has been provided to help you test some of the functionality of your hash table and answer the questions. It takes six command-line arguments:

1. the number of items to hash
2. the seed for the random number generator

3. the maximum load factor (as a real number, not a percent, e.g. 1.25 not 125)
4. the initial size of the hashtable
5. the maximum chain length (use 0 to indicate there is no maximum chain length)
6. the name of the file for output

The sample output was obtained by running `TestHash` with the command line arguments "`5 292929 0.8 2 0 sampleOutput.txt`". You can change this code as you wish to test your entire `HashTable` class.

Summary of provided materials

- `HashTable.java` - you will need to modify this class
- `Questions.txt` - you will need to modify this file
- `TestHash.java` - you can modify this class

How to proceed

After you have read this program page and given thought to the problem we suggest the following steps:

1. Look over the Javadoc documentation for the `HashTable` and `LinkedList` classes.

2. Review these style and commenting standards that are used to evaluate your program's style.
3. Download the provided files to your program 4 development directory.

4. Incrementally implement and test `HashTable`. One approach would be to first implement the hashtable ignoring load factor and chain length issues, then add in the ability to resize if the maximum load factor is exceeded, and then add in the ability to resize if the maximum chain length is exceeded.

   Your code will be much simpler to write and easier to debug if you make sure your hashtable is an array of `LinkedLists` and initialize each index to be a new `LinkedList` (in the hashtable constructor). Recall that you can't instantiate an array containing items that include a generic type, e.g.,

   ```
   LinkedList<E>[] arr = new LinkedList<E>[10];  // WON'T COMPILE
   ```

   but you can instantiate without a generic (which means the generic defaults to `Object`) and cast to the generic type, e.g.,

   ```
   LinkedList<E>[] arr = (LinkedList<E>[])(new LinkedList[10]);
   ```

   When you do this, your compiler may give you a warning about unsafe or unchecked operations but it is ok to ignore the warning.

5. Answer the questions in Questions.txt. This will involve running `TestHash` several times and analyzing your results.

6. Submit your work for grading.

Handing in

What should be handed in?

Make sure your code follows the style and commenting standards used in CS 302 and CS 367.

Electronically submit the following files to the Program 4 tab on Canvas:

- "`HashTable.java`" containing your modified `HashTable` class
- "`Questions.txt`" containing your answers to these questions

Please turn in only the file named above. Extra files clutter up the "handin" directories.

Last Updated: 1/11/2018