Homework # 11

due Monday, April 23, 10:00 PM

In this homework, you will start a multi-week project to identify duplicate resources on a
website. This week, we will define a well-known hash of content (CRC-32) and a hash table using
URI’s as keys.

1 CRC-32

A “cyclic redundancy check” is a way to compute a check sum of a file which is unlikely to remain
the same if there are inadvertent changes. Technically, this process is done by polynomial division
over the field defined for \{0, 1\}. But you won’t need to do anything so complex. Instead use the
Wikipedia article and a MSDN web page linked from the Wikipedia article. Please don’t use other
resources—in particular don’t look up a Java implementation. This assignment should give you
practice in implementing documented algorithms, and with using bit operators in Java:

xor Use \(a \oplus b\) to XOR the values in \(a\) and \(b\)

and Use \(a \& b\) to AND the values in \(a\) and \(b\)

shr Use \(a \gg \text{b}\) to shift right the (unsigned) value in \(a\) by the amount in \(b\). (The related \(\gg\)
operator does \textit{signed} right shift.)

We will be computing CRC-32 over an input stream which is a source of bytes. See the Javadoc
for \texttt{InputStream} for how to use it.

2 Concerning the Resource ADT

For this assignment, you will implement a special-purpose hash table that maps URI (Uniform
Resource Identifiers) to “Resource” objects. The class \texttt{Resource} is abstract and in this assignment
has two concrete subclasses \texttt{FixedObject} and \texttt{URLObject}. The former is created with an object.
The latter include a URL, and the CRC-32 hash of the contents of the resource located at the URL.

3 Concerning Hash Tables

The hash tables for this assignment will use the idea of “bucket hashing” (also known as “chained”
hashing). See the textbook pages 600f (3rd ed., pp. 588f). In bucket hashing, the table puts all the
elements that hash to a particular array index into a linked list.

Our hash table is nonstandard in that it is an “endogenous” table; each \texttt{Resource} object has
a “next” field that is used to link up the resources mapped to the same bucket. As a result, a
resource can only legally be added to one table, which is not a problem since we only expect to
have a single table.

Unlike the textbook, our hash table will increase in size when the table has too many elements
in it. The table should initially have an array of size 7. When the number of elements in the set
becomes greater than the capacity of the array, a new array should be allocated and all items should
be “rehashed.” The new array should have size \(2^n\), where \(n\) is the current number of elements.

Once the new array is allocated, all entries need to be placed in their correct places, which
are possibly different than before because the array has changed size. Because the nodes are
endogenous, we cannot swap data when rehashing. Therefore, we will need to rearrange all of the references to move the nodes from one table to another.

4 Implementing the Resource.Table ADT

This ADT functions as a Set ADT (as seen in Homework #9, but specifically for “Resource”), but also includes the get method (of the Table ADT), but not put, since every resource already includes its own URI. The get method takes a URI and returns the resource in the table under that URI, or null if no such resource exists.

The fact that the table is keyed on the URI but contains resources means there are some qualifications to the expected semantics for Set methods:

- add This takes a resource, but if a resource with this URI already exists in the table, it is removed and replaced with the parameter resource (of course, if the resource itself is already in the table, no change should happen). The add method returns true only if the end result has more elements than previously. (In other words, replacement causes the return value to be false.)

- contains This method take an object; it returns true if a resource in the table is equal to the parameter. Since we make sure that two resources are equal only if they have the same URI, this means you can implement this efficiently by looking up the URI in the parameter resource.

- remove Similarly, we only remove a resource if it is equal to the parameter.

The class has an object invariant. You should be able to figure this out from the “DESIGN” given at the top of the Resource.Table class. We provide test cases for this invariant.

As with all Set classes, this class provides an iterator over the elements of the table. This should be done simply by going over the array and then through each chain of elements. As with some previous implementations, we will have current and next pointers, which are equal when there's no element to remove. We provide the invariant checker for the iterator for you.

5 Files

The repository for this homework assignment includes the following files:

- src/TestInternals.java Tests for the invariant checker.
- src/TestCRC32.java Tests for the CRC32 class.
- src/TestResourceTable.java Tests for the Resource.Table class.
- src/TestEfficiency.java Efficiency tests for the Resource.Table class. (Should take under 20s.)
- src/UnlockTests.java Unlock all locked tests.
- src/edu/uwm/cs351/CRC32.java Skeleton file for the CRC-32 class.
- src/edu/uwm/cs351/Resource.java The Resource class, including a skeleton of the Resource.Table class.

As with previous assignments, we provide random testing in the JAR file.