

Computer Science 62

Lab 6 (updated)

October 14, 2015

It's back! Yes, `GridTest` and the `CompressedTable` class are back again to torment you. We had you write a `CompressedTable` class for Assignment 4, but neglected having you write iterators for the table.

As we've seen for trees, there are sometimes multiple ways to iterate through collections. For tables, it makes sense to iterate through them a row at a time (so-called row-major order) or a column at a time (column-major order). For example, suppose we have an array `elt` with two rows and 3 columns. A row-major iteration would give us the elements in the order: `elt[0][0]`, `elt[0][1]`, `elt[0][2]`, `elt[1][0]`, `elt[1][1]`, `elt[1][2]`. On the other hand a column-major iterations would produce them in the following order: `elt[0][0]`, `elt[1][0]`, `elt[0][1]`, `elt[1][1]`, `elt[0][2]`, `elt[1][2]`.

Of course in this program we are working with `CompressedTable`, not an array. On the other hand, we have a method `getInfo(r,c)` that retrieves the element in row `r` and column `c`.

For this lab you are to add an iterator: `rowMajorOrderIterator` to the class `CompressedTable`. We have provided you with starter code that includes a correct implementation of the complete `CompressedGrid` program.

To see how to accomplish this we suggest that you review the implementations of iterators in the Bailey's `structure5` library. While you could write the iterator class `RowMajorIterator` as a separate class from `CompressedTable`, we instead recommend that you write it as a private "inner classes". That is, you write the class definition inside the `CompressedTable` class. One advantage of this is that you get access to the protected instance variables of the `CompressedTable` class, e.g., `numRows` and `numCols`. The iterator methods simply create an iterator from these classes and return it.

In the `main` method of `CompressedTable` we include code to build a compressed table as well as (commented out) code to iterate through it in row-major orders.

Big-“O” complexity

We assume that you used the method `getInfo` from `CompressedTable` to get and return the successive elements in the iterator you wrote. What is the complexity of `getInfo` (in big-“O”

terms when n is the number of entries in the table)? What is the complexity of the complete traversal using your iterator?

Please include your answers to these questions in a comment at the top of your `CompressedTable` class.

Java 8 Version

We would also like you to write a method to perform a row-major traversal of the compressed table, performing an action on every element of the list. The header should be as below:

```
public void doInRowMajorOrder(Consumer<ValueType> action) {
```

Once you get this working properly you should be able to get it to print out all the elements of a compressed table `tab` in row-major order by writing:

```
tab.doInRowMajorOrder(v -> System.out.println(v));
```

Extra Credit

Your iterator implementation likely modifies the state of the `CurDoublyLinkedList` (think about the `current` variable) in `CompressedTable`. This means that it is not possible to have two iterators iterating through the table independently. For extra credit, rewrite your iterator such that it does not modify the state of `CurDoublyLinkedList` or any other state in `CompressedTable`. Additionally, ensure that the total time for a complete iteration is still $O(n)$. To do this you will need to directly access the contained information and store any necessary state in the iterator itself. This is a bit tricky to write and debug so do this only if you have finished your other work for this class.

Turning it in

Drop your lab off in the dropbox as usual. Be sure to name it properly.