Compression

Assignment 4

Due Sunday, October 1, 2017

CSC 062: Fall, 2017

Objectives

For this assignment, you will:

- Gain more experience using JUnit.
- Practice implementing and using a doubly linked list data structure.
- Gain experience designing and implementing a non-trivial algorithm.

Compression

Sometimes we need to store massive amounts of information about an object. A good example is storing graphic images. To save space on disks and in transmission of information across the internet, researchers have designed algorithms to compress data. In this assignment you will learn one of these compression techniques.

A graphic image can be represented by a two dimensional array of information about the colors of various picture elements (or pixels). At high resolution the image may be composed of 1000 rows and 1000 columns of information, leading to the need to store information on 1,000,000 pixels per image. Needless to say this creates serious problems for storing and transmitting these images. However most images tend to have many contiguous groups of pixels, each of which are the same color. We can take advantage of this by trying to encode information about the entire block in a relatively efficient manner.

The basic idea of our encoding will be to represent a block of pixels with the same color by simply recording the first place where we encounter the new color and only recording information when we see a new color. For instance suppose we have the following table of information (where we will imagine each number represents a color):

2	2	2	3	3
2	3	3	3	3
3	1	1	1	3

If we imagine tracing through the table from left to right starting with the top row and going through successive rows then we notice that we only need to record the following entries:

2	-	-	3	-
2	3	-	-	-
-	1	-	-	3

Rather than recording this in a two-dimensional table, it will now generally be more efficient to keep this information in a linear list of `Assocation`s where it is assumed we sweep across an entire row before going on to the next:

$(0,0) \to 2$	$(0,3) \to 3$	$(1,0) \to 2$	$(1,1) \to 3$	$(2,1) \to 1$	$(2,4) \to 3$

This assignment asks you to design a class which will represent one of these compressed tables. A working demo of this program can be found at
`http://www.cs.pomona.edu/classes/cs062/assignments/demos/CompressedGrid/CompressedGrid.html`
It will only run if you enable Java for this page on your browser.

We have provided you with a lot of code here, but you will find that much of the code you must write is quite tricky. This project will require you to be very careful in developing the code for the methods. Look carefully at the provided code and design your methods very carefully. In particular, be sure to test your code carefully as it is developed as you will likely make several logical errors if you are not extremely careful.

This is your most complex program yet. You should start early on this assignment and make a very complete design for your program before you ever sit down at the computer to program.

Classes

CurDoublyLinkedList class

Class `CurDoublyLinkedList` extends `DoublyLinkedList` from Bailey's `structure5` package. (You can find the code for DoublyLinkedList under the Bailey Structure5 source code link on the Documentation and Handouts page of the course website). Think carefully about what it means for one class to extend another.

The `CurDoublyLinkedList` class should support all of the old methods of the `List` interface. In addition the new class should support the following methods:

- `first()`, `last()`,
- `next()`, `back()`,
- `isOffRight()`, `isOffLeft()`,
- `currentValue()`,
- `addAfterCurrent(Object value)`, and `deleteCurrent()`.

Specifications for these methods can be found in the startup code available on-line. You should start this assignment by finishing the `CurDoublyLinkedList` class.

TestCurDoublyLinkedList class

This is a **JUnit** test class for the `CurDoublyLinkedList` class. There are already a few tests provided for you. You must finish this class by adding at least one test for each method in the `CurDoublyLinkedList` class. The more thorough your tests, the easier time you will have when you implement the `CompressedTable` class. Be sure to test all of the *edge* (special) cases.

CompressedTable class

The `CompressedTable` class represents the compressed table. `CompressedTable` implements the `TwoDTable` interface. It has an instance variable `tableInfo` of type:

```
CurDoublyLinkedList<Association<RowOrderedPosn, ValueType>>
```

The instance variable `tableInfo` is a `CurDoublyLinkedList` where each node in the list is an `Association` whose key is an entry in the table of type `RowOrderedPosn`and whose value is of type `ValueType`. Feel free to add other instance variables to this class.

You must fill in the constructor for the CompressedTable class as well as the two methods:

- updateInfo(row,col,newInfo)
- getInfo(row,col)

The `updateInfo` method of `CompressedTable` is probably the trickiest code to write. Here is a brief outline of the logic.

1. We have provided you with code to find the node of the list that encodes the position being updated. Of course not every position is in the list, only those representing changes to the array. If the node is not there, the method returns the node before the given position in the list. The class `RowOrderedPosn` (see the startup code) not only encodes a position, but, because it also contains information on the number of rows and columns in the table, can determine if one position would come before or after another.

2. If the new information in the table is the same as that in the node found in step 1, then nothing needs to be done. Otherwise determine if the node represents exactly the position being updated.

   If it is the same, update the value of the node, otherwise add a new node representing the new position.

3. If you are not careful you may accidentally change several positions in the table to the new value. Avoid this by considering putting in a new node representing the position immediately after the position with the new value. (Why? Draw pictures of the list so you can see what is happening!)

4. If there is already a node with this successor position then nothing needs to be done. Otherwise add a new node with the successor position and the original value. (Do you see why this is necessary? Look at the demo program to see why.)

Try to draw examples of this logic with several sample lists so that you can understand how it works!

RowOrderedPosn class

The `RowOrderedPosn` class represents a single entry in a row-ordered table. The constructor takes four parameters: the row of the entry in the table, the column of the entry in the table, the total number of rows in the table, and the total number of cols in the table. Thus,

```
new RowOrderedPosn(0, 0, 5, 3)
```

represents the entry at location (0, 0) i.e. the upper-left corner in a table with 5 rows and 3 columns. This class also contains methods to return the next position after a given one and to compare two positions in a table. This class is already implemented for you.

DrawingPanel class

This class is responsible for displaying the two-dimensional grid of colored rectangles. It is also responsible for any mouse actions performed on the two-dimensional grid. This class is already implemented for you.

GridTest class

This class creates an applet that lets the user manipulate a grid of rectangles that form an image. The user interacts with the application by clicking on a color button to set the current color and then clicking on rectangles in the grid to change the colors of individual rectangles. Along with the color buttons there is a button that will display the results of sending the toString method to the object of type `CompressedTable` to show the current state of the representation. This can help you as you attempt to debug your `CurDoublyLinkedList` and `CompressedTable` classes. This class is already implemented for you.

TwoDTable interface

This interface represents a two-dimensional table.

Getting Started

1. Read through this writeup completely before you start. Then, get a sheet of paper and pencil and draw pictures to help you understand how the doubly linked list works and how the compressed table should work. These examples can also help you form your unit tests. Don't forget to think about special cases.

2. After reading the writeup and going through examples, start working on the design of the program. How will you keep track of whether current is off the right or left side of the list? Look out for methods that you can implement in terms of the other existing methods in either the `CurDoublyLinkedList` or `DoublyLinkedList` class.

3. Create a new project in Eclipse and copying the starter files from `/common/cs/cs062/assignments/assignment04/` into the `src` directory of your newly created project.

4. Try to interweave testing your code and writing your code. It is much better to write a method and then stop and test it instead of writing all of the code for a class and only afterwards testing. Even better is to write all of your test in JUnit before you write the code and then slowly turn the red to green!

5. To ensure compatibility with the auto-grader, update the build path of the project and include `AutograderCompTest.jar` by selecting the menu Project → Properties → Java Build Path → Libraries → Add JARs. Initialize an instance of `AutograderCompTest` in a main method and call `testCurDoublyLinkedList()` or `testCompressedTable()`. Note that this test class only checks compatibility, not correctness.

Grading

You will be graded based on the following criteria:

criterion	points
No change if new color same as current	1
Change color of position already in list	2
Change color of position not in list	2
Correctly adds second node to list when needed	2
`CurDoublyLinkedList`	4
JUnit tests for all methods in CurDoublyLinkedList	2
General correctness	2
Appropriate comments (including JavaDoc)	2
Style and formatting	2
Submitted correctly	1
extra credit - design	2
extra credit - efficiency	2

NOTE: Code that does not compile will not be accepted! Make sure that your code compiles before submitting it.

Extra credit

Design

This is your most complex program yet. As a result you should make a very complete design for your program before you sit down at a computer to program. If you email your design for the methods `removeFirst`, `removeLast`, and `removeCurrent` of `CurDoublyLinkedList` and `updateInfo` of `CompressedTable` to `kim@cs.pomona.edu` by Thursday at midnight, we will take a look at them and provide you with feedback on them. (Keep in mind that methods in `CurDoublyLinkedList` can make calls to `super` to access methods in `DoublyLinkedList` and then add your own behavior.)

While these should not be written in Java, you should include the complete logic of the methods. Remember that you must draw pictures and look at all possible special cases in order to get these right. You can get up to two points extra credit for submitting these via e-mail by the deadline. Just paste your design into the email message.

Space Efficiency

As you add more information to the table, you will notice that the table is no longer as efficient in space, because several consecutive entries may have the same values. Make the representation more efficient by dropping later values if they can be subsumed by earlier ones.

For example, the list

$(0,0) \to 2$	$(0,3) \to 3$	$(1,0) \to 3$	$(1,1) \to 3$	$(2,1) \to 1$	$(2,4) \to 3$

can be replaced by the much simpler list:

$(0,0) \to 2$	$(0,3) \to 3$	$(2,1) \to 1$	$(2,4) \to 3$

For extra credit, modify the `updateInfo` method of `CompressedTable` to eliminate consecutive items with the same value. The amount of extra credit received will be proportional to the efficiency of your algorithm. Ideally this optimization will only take O(1) time each time something is inserted in the table.

Submitting Your Work

Export your project from Eclipse and submit it as usual using the instructions at `http://www.cs.pomona.edu/classes/cs062/handouts/eclipse_submission.pdf`. Please follow these very carefully. If you do not, your program will NOT be submitted correctly and you will not receive credit for your assignment (and you and we will all be very sad!). Dont forget to fill out the asg04.json file! In particular, dont forget to set the "ec" field to true if you did extra credit (beyond the design).