CS780/880 Programming Assignment 1

September 10, 2014

Due: Wednesday, September 17 at midnight. (Submission details will follow.)

This assignment is intended to provide you a vehicle for learning and experimenting with some of the basic features of the Granite system. The goal is for you to modify and extend some Granite file utilities.

Startup: download files from *cs.unh.edu/~rdb/cs880/granite*

1. ***granite.jar***: You should **not** need to un-jar this file, but you must put it in some "standard" location (like in you a *lib* file in your file system home. You must add the jar file at this location to your CLASSPATH. *Caveat*: Last year, some people using Windows could only get this to work by un-jarring granite.jar and putting the resulting directory into their CLASSPATH. I no longer remember whether this problem was connected to whatever IDE they were using. Try it first as a simple *jar* file.
2. ***SimpleStats.java*:** This is a simple program that reads any Granite datasource and computes and prints the min and max values of all attributes in the file. It does this by reading each *datum* for every location in the domain (array) that is defined by the datasource. You will be editing this file.
3. ***BasicDemo.java*:** This is a rather complicated, but still basic, Granite demo that uses multiple different data access methods to compute the same results as *SimpleStats*.
4. ***data.tar.gz*:** A tar ball of a variety of mostly small data sets with different attributes. Some are 2D; some are 3D. Many have multiple attributes. There is also one "real" data set: a version of an MRI image of a head.

Tasks

1. Extend *SimpleStats.java* to also compute min/max using the *minMaxByAllInPointOrder* method in *BasicDemo*. You may just copy it into your solution – and make sure that it works by adding a call to it right after the existing call to *minMaxByDatumThenAttribute.* This is the most efficient option in most cases, but it breaks down for very large data sets since it requires that all data be stored in primary memory at once.
2. Create a third method for computing min/max values that you can call *minMaxByChunk*. This approach will by a variation on *minMaxByAllInPointOrder*. Instead of reading the entire data set at once, you will read it one slice at a time. This approach only makes sense for 3D data sets: *demo, demoBig*, *head64,* and *head-float-w1*. You may simply return with an error message if you are called with a datasource that is not 3D. See details below about this method. Add a call to this method after previous *minMax* calls.
3. Surround the 3 calls to *minMax* methods with the simple Java timing code as used in *BasicDemo* in its *minMaxTests* method. Don't copy that method's structure, just insert copies of the timing code and printouts between your calls.

minMaxByChunk

1. This version must read 1 slice at a time from the 3D data set using the *subblock* Granite method and update the min/max values based on each slice.
2. Keep the method simple; define other methods to share code and to keep any one method from getting too large.