In this program you will code up your first major CAD algorithm within Aphyds – a bipartitioning algorithm based upon Fidducia-Mattheyses. This algorithm seeks to minimize the number of nets connected to nodes in both partitions (the “cutsizes”).

**Getting Started**

You will build upon the same code base as homework #1. Again, to run the program you compile and run “CircuitViewer.java”. Then, open a circuit (File->open) and start partitioning (Edit->Partition Circuit). Notice that the main circuit view has changed, with a horizontal red line added and many nodes shifted. The circuit has been randomly partitioned, with partition 0 on the top, 1 on the bottom, and the cut line shown in red. Click on a node to bring up the node dialog, and notice that the Partition box has been filled in. Change the partition of this node (0 to 1 or 1 to 0), and notice that the main viewer and the partitioning dialog box changes. Feel free to try to manually get a good partitioning of a circuit. Notice that if you move too many nodes into a given partition, it’s size in the partitioner dialog box turns red. This indicates a partitioning that violates size constraints.

Any time you want to re-initialize the circuit, type in a number in the seed box on the partitioner dialog box and hit enter. Each number produces a random initialization, though each number always produces the same initialization. This will be useful during testing.

**Programming Assignment**

In this programming assignment you will need to implement the partitionCircuit and computeNetGains routines in Partitioner.java. Again, you should look at the Partitioner.html page produced by Javadoc to understand what needs to be done, and what routines will be useful in your efforts. You should implement a Fidducia-Mattheyses partitioner.

Two data structures are provided to help you in this process. Bucket.java implements the gain buckets used by the FM algorithm. Specifically, you can efficiently insert and remove nodes from this structure. You can also ask the structure for the node with the highest gain currently within this structure.

PartitionTranscript.java keeps a running transcript of the partitioning process. It is useful for providing a graphical display of the results when you are done. It also helps you to roll back the last N moves in the partitioning, something you will need to do in your FM algorithm (remember, FM moves as many nodes as possible, then rolls back to the best state found within that pass).

One last very useful routine is in Partitioner.java: printStatus(). This routine dumps the status of most of the data structures to the screen in a readable format. You’ll probably insert calls to printStatus at multiple places in your code as you are debugging, and perhaps edit the routine to show you what’s most important to you. When you are done, and have a running algorithm, remember to remove all calls to this routine.

To run your program, open the partitioner dialog box (described above), and then push the button. It will partition from the current partitioning. Thus, the previous random initialization, plus any moves you’ve made via previous partitionings and/or via the node dialog box will form the starting state for the algorithm. If you want to try another initialization (or reset the initialization) you should change the seed in the partitioner dialog box.

This programming assignment will require understanding the public and protected functions (those that appear in the .html files from Javadoc) for Partitioner.java, PartitionTranscript.java, and Bucket.java, plus those you learned in the previous assignment. You should only need to actually modify partitionCircuit() and computeNetGains().
Experiments

Once your algorithm is running, you should take a look at the partitioner graph. This shows the cutsizes (vertically) vs. iteration (horizontal). The yellow bars indicate the ends of a pass, and the green circle is the partitioning that was accepted during a pass. This graph should give you a good understanding of how this algorithm works overall.

You should also try multiple partitionings of a given circuit. Give it various seeds, then run the partitioner. The results will be stored in the list in the partitioner dialog box. Take a look at how the algorithm changes results for different seeds.