To finish off the routing from program #5, we now have to do detailed routing. In this program you will implement the basic left-edge algorithm.

Algorithm Description

In order to use the left-edge algorithm, you must first create the HCG (Horizontal Constraint Graph) and the VCG (Vertical Constraint Graph). The HCG captures information about whether two nets overlap horizontally, and thus cannot occupy the same track. Specifically, if a connection of one net is in-between connections of the other net, then they overlap horizontally. An edge would thus be added to the HCG, indicating that these two nets must be in separate tracks.

The VCG helps determine the order vertically that nets must be placed. Specifically, consider two nets that have connections in the same column of a track, with net1 connected to the top edge, and net2 connected to the bottom. If net2 was assigned to a track higher than net1, the branches for the two nets would overlap in this column, causing a short-circuit. Thus, in this case a link is added to the VCG from net1 to net2, indicating that net1 must be placed above net2.

One important observation about the VCG is that for some channel routing problems the VCG will contain a cycle, meaning that the left-edge algorithm cannot be used. We could force you to write a more complex algorithm to handle these cases (and many do exist). However, we instead cheat – before the VCG creation algorithm the Aphyds system does a quick check to see if there will be a cycle in the VCG. If there will be, Aphyds chooses some top connections to route in on polysilicon instead of metal2. This means that the top and bottom branches can overlap without causing a short, since they are in different levels. Thus, the Aphyds system removes all potential cycles, allowing your code to just handle situations where the left-edge algorithm works.

Once you have the HCG and VCG, you can perform the left-edge algorithm. This algorithm works from the top track (track 0) to the bottom track (track I is below track I-1). We assign to a track the leftmost unrouted net that has no predecessors in the VCG (indicating there are no nets that must be placed above it). Additional nets will be added to this track if possible, again choosing the leftmost net with no predecessors in the VCG, and no conflicts from the HCG. This process continues to the next track down, until all nets have been assigned tracks. Note that when a net is placed it is removed from the VCG, removing the restriction on it’s children and allowing them to potentially be assigned.

Your programming efforts will focus on the function computeCGs and channel_router in Channel.java. computeCGs generates the HCG and VCG. channel_router does the actual routing of the channel, using the left-edge algorithm discussed above. Your job in this programming assignment will be to write these routines.

Getting Started

You will build upon the code you produced in the previous programming assignments. Again, to run the program you compile and run “CircuitViewer.java”. Then, open a circuit (File->open) and start placement (Edit->Channel Route Circuit). Note that to do detailed routing you do not need to do placement or routing first – in version 1.0 of Aphyds global and detailed routing are not yet linked, and the channels you route are randomly generated.

The channel router dialog box shows the channels to be routed at left. The tracks are shown vertically, with track 0 at the top. The columns are shown horizontally, with 0 at the left. Above and below the tracks is shown the net number being routed. The job of the channel router is to link up all locations with the same number by a net in a single track, while seeking to minimize the total number of tracks required.

Once your detailed router is running, the results will be displayed at left. To aid in checking your results, you can click on any part of a net and its entire routing will be highlighted in white. Thus, you can click on one of the net...
labels (the numbers at top and bottom) and see if all other labels with the same number are highlighted, yet none of the other numbers are.

A single chip layout will have multiple channels. To handle this, the list at right indicates the various channels in your circuit. By clicking on a given channel you load that channel into the viewer. Then, clicking on the “Route Channel” button calls your channel router. The resulting routing will be displayed, and the entry in the list at right will be augmented with the track count for that channel. Note that, due to higher routing congestion, the channels near the center of the chip will have more nets to route than those near the edges.

**Programming Assignment**

In this programming assignment you will need to implement the functions computeCGs and channel_router in Channel.java. You should look at the Channel.html page produced by Javadoc to understand what needs to be done, and what routines will be useful in your efforts.

One diagnostic routine is printCGs. At the end of computeCGs it prints out the status of both the VCG and HCG, allowing you to check whether the graphs computed actually reflect the true requirements of the channel.

As is shown in the Channel Router dialog box, the main input to your algorithm is a list of the connections to the top and bottom of the channel. These are provided to you in top_connections[] and bottom_connections[]. If top_connections[I] or bottom_connections[I] is 0, this means no net is connected to that side of the channel in column I. A positive number N indicates that the Nth net is connected to that location in top_connections[], a value of –N indicates that the Nth net is connected, and it comes in on polysilicon. As discussed earlier, this is a simplifying assumption for educational purposes. The main issue is that all locations with either N or –N stored in top_connections[] and bottom_connections[] must be connected together by a single net, in a single track. The only impact of the negative value in top_connections[] is that, when you are creating the VCG, you should not add a link because of a column with a negative net value. Since the top is in polysilicon, and the bottom in metal2, they can overlap without any conflict.

This programming assignment will require understanding the public and protected functions (those that appear in the .html files from Javadoc) for Channel.java. You should only need to actually modify computeCGs() and channel_router().

**Experiments**

Once your algorithm is running, you should check the results of some example channels. Most importantly, you should highlight each number in the channel by clicking on it. Make sure that all values with the same number are highlighted simultaneously, and no other numbers are highlighted. If this doesn’t happen there is a bug in your code. Also, you can examine the routing visually and see whether it is a reasonable routing for this channel, or if your implementation of the left-edge algorithm is incorrectly assigning tracks.