1 Parsing

Parsing the data from the .dot file was surprisingly difficult. The underlying procedure was understood, as I was just copying what I did in lab 12. I I used *fgets* to read each line of the .dot file. I then scanned through each character of the line and extracted any numbers. I used *popen* combined with *grep* to pick out the specific lines needed at various times throughout the code. For example the I popened grep to identify all lines with "R = " to pick out the lines containing the resistances. I used the -r flag with grep to identify how many resistors where in the circuit. I then used this number of resistors to allocate memory for the Gauss-Jordan matrix (which I called circuit), the array of resistor positions and resistances (which I called resistors), and the Gauss-Jordan answer array (termed I).

For whatever reason invoking malloc was not working and the code kept reporting with either memory errors or seg faults. I worked around this by defining the above three matrices later in the code with a direct memory allocation, no pointers. I also had to be very specific about where the the data from the file was stored in any of the matrices. This reduced the modularity of my code, but not to a level that limited its functionality.

2 Solving

Once the data from the file was parsed and stored in the Gauss-Jordan matrix I could solve it. To do this I copied my Gauss-Jordan code from lab 11. Due to the way I wrote that lab very little modifications, other than variable names, needed to be included.

3 Loops and Complexity

There are twenty-one loops in total in this code. The first loop ensures that the array that stores numbers from the file is empty. This shouldn't be necessary, but was implemented to avoid memory errors. It has no loop invariants.

The next three loops read the location of the resistors and their resistances into a matrix. The first loop runs through each relevant line of the file. The second loop runs through each character in the line. The third loop resets the the array that stores the numbers. The first and third loops here have no invariants. The only invariant in the second loop is the buffer that stores the line of the file from the first loop. Loops five to seven follow the same format, but for voltage, and have the same invariant as these three loops. Loops thirteen to fifteen are the same (including invariant) but for identifying the Kirchhoff loops. Loop sixteen, which is part of loops thirteen to fifteen, stores the resistances of the Kirchhoff loop into the Gauss-Jordan matrix (named circuit in my code). The loop invariants in loop sixteen are the nodes, a and b, as well as the array of resistor positions and resistances.

Loops eight and nine store the current equations in the Gauss-Jordan (circuit) matrix. The loop invariant here is the array of resistor positions and resistances. Loop ten stores the answers to the current equations (zero) in the Gauss-Jordan matrix. This loop has no constants. Loops eleven and twelve store the answers to the voltage equations in the Gauss-Jordan matrix. The loop invariant in these two is the voltage across the battery.

Loops seventeen through nineteen solve the Gauss-Jordan matrix. Loop seventeen runs through each column of the Gauss-Jordan matrix. Loop eighteen runs through each row of the Gauss-Jordan matrix. Loop nineteen runs through each element in the Gauss-Jordan row. In all three loops there are no constants. Loop twenty finishes solving the Gauss-Jordan by dividing the answer by the diagonalized matrix elements. The loop invariant here is the diagonalized Gauss-Jordan matrix. Loop twenty-one finished the code by exporting the results to a file. The loop invariants here are the array of resistor positions and the array of currents.

The most time consuming portion of this code is solving the Gauss-Jordan. Since the matrix is (almost) square, the number of runs for each of the three Gauss-Jordan loops is equal to the number of resistors. As such the code is dependent on the cube of the resistor numbers. This gives a big O complexity of $O(n^3)$ with a cubic run time.