Krisha Sharma

CSE 13s

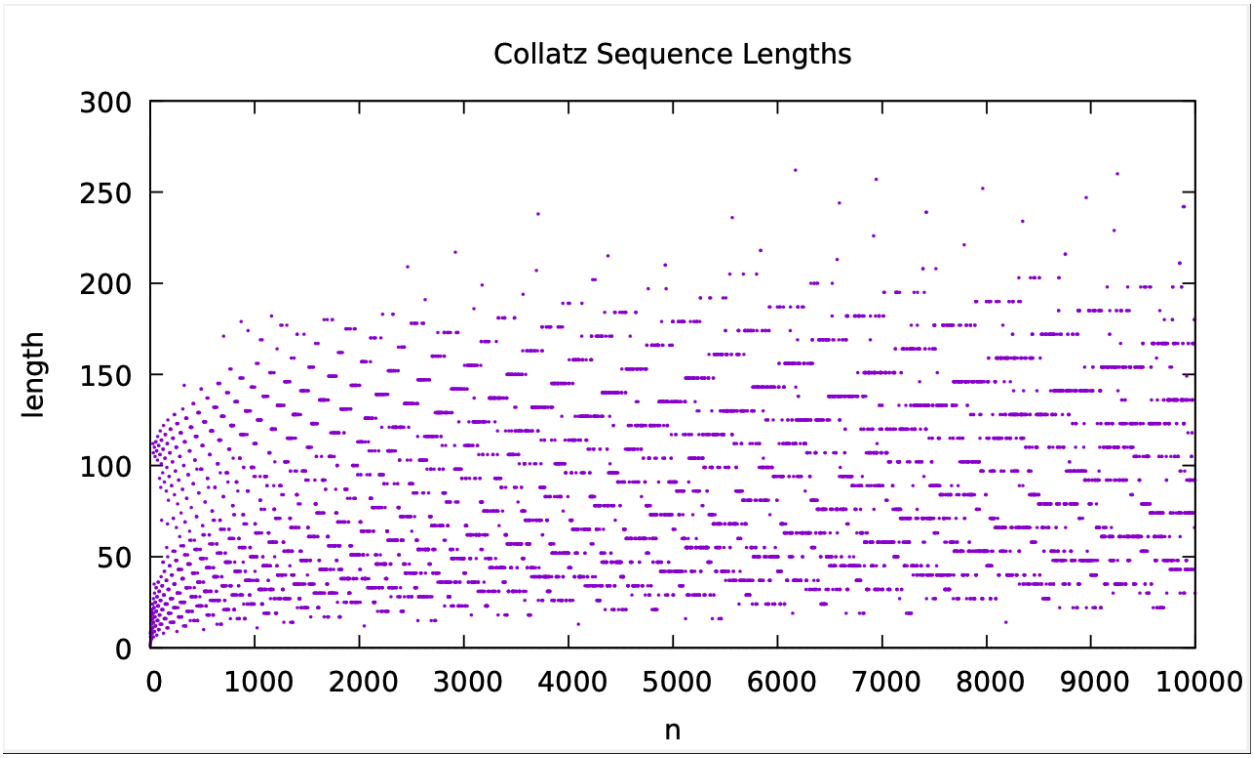
Winter 2022 Long

Assignment 1: Getting acquainted with UNIX and C Write Up

Introduction:

Our goal in Assignment 1 was to get familiar with a UNIX system. The objective was to write a bash script that produced plots of a provided C program. The C program that was provided printed Collatz sequences that started from some positive integer n. Overall, during this assignment, using the provided collatz program, we were expected to familiarize ourselves with the syntax of C, understand what the provided program was doing, and finally produce interesting plots using the collatz sequences produced by the provided program. It was required to use gnuplot to produce the plots and everything was to be automated by means of a bash script. Gnuplot is a utility that can be run on a variety of platforms to generate high quality graphs and plots; however it is used through the command-line, the shell. Gnuplot is mostly used to plot data points in a file and for this assignment it was what we used to generate the three example plots given in the assignment 1 document.

Collatz Sequence Lengths (Figure 1):



The Collatz Sequence Lengths graph is the first figure that we were assigned to replicate in assignment 1. We were given the task to reproduce this graph using gnuplot and bash scripting, more specifically shell scripting. Overall this graph was fairly easy to replicate and understand due to the TA's section. Eugene held a section on January 7th, 2022 and during it he walked students through how to start the assignment as well as the important concepts and ideas that we will need to know in order to successfully replicate the other two plots. The figure itself is a graphical representation of all the sequence lengths given by the collatz program. The graph goes from n in the range from two to 10 thousand, and for each one, it plots how long the sequence is. The length of a sequence is just how many numbers are in that sequence. The collatz.c program provided only prints the Collatz sequence starting from some positive integer n.

In order to create a graph that shows all the collatz sequence lengths it was necessary to use a for loop with a specified range. The range in the for loop naturally reflects the range given in the graph, in this case it was 1 to 10,000.

To start to generate the plot it was necessary to place all of the given collatz sequences into a file. By doing so it makes it easier to generate the line count of the file and thus see exactly how many lines were generated giving us the length of the sequence. By piping the output of the collatz sequence in the length.dat file, I was able to see the length of the sequence.

```
./collatz -n $n | wc -l >> /tmp/length.dat
```

The shell command above gets the line count of a collatz sequence and appends it to the length.dat file specified. The wc command looks at the word, line, character, and byte count of the file but the -l specifies that I want the number of lines. Because the collatz sequence length varies and changes, it is necessary to clear the length.dat file each time the shell script is run.

```
rm -f /tmp/length.dat
```

This specific bash command forcibly clears out my file titled length.dat. Temporary files in a linux environment are world-writable adn world readable, meaning that any user in the system can read and write to the temporary directory. In most linux systems “/tmp” directory is used as a temporary directory and any user or processes in the system can use this directory to store any temporary data. In this case I was using a temporary directory to store my length.dat file which would be forcibly cleared every time I ran my shell script plot.sh.

```
echo -n $n \ >> /tmp/length.dat
```

At this time /tmp/length.dat only has the sequence length. In order to get an x coordinate to complete the graph, it was necessary to echo x and place everything on the same line into a

length.dat file for gnuplot to read. Echo x naturally creates a new line; however, it is possible to not print the trailing new line character by adding -n.

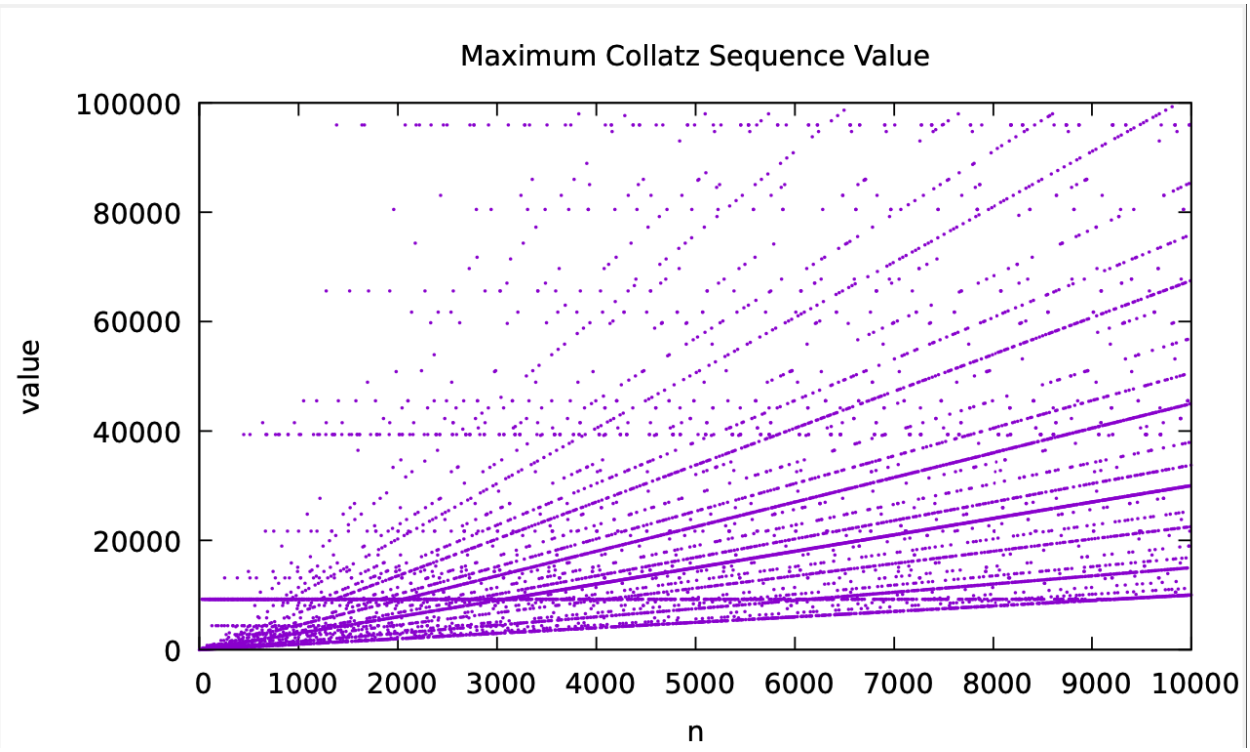
All of this will be placed within a for loop that has a range of 1 to 10,000 as the x axis goes to 10,000 in the figure provided in the assignment 1 document.

In the gnuplot commands it was important to change the output to length.pdf, the title to sequence lengths and the plot location to length.dat with points. For figure 1, it was also important to take the existing program and format the output to what was needed; that way gnuplot is able to plot out the sequence lengths derived from the collatz sequence which is given.

Collatz Sequence Lengths (Figure 1) Thoughts:

Overall, I didn't run into any problems trying to make this figure as Eugune basically walked me through how to do it during his section on the 7th. I did have to adjust the range myself and play around with the labels for the x and y axis, but other than that the figure was super straightforward and easy to make.

Maximum Collatz Sequence Value (Figure 2):



The second figure I had to make for assignment 1 was the Maximum Collatz Sequence Value graph. This graph is a representation of the maximum values for collatz sequences in the range of 2 to 10,000. The shell script to create this graph is almost identical to the shell script used to generate the first figure, as both of them are very similar. In order to create this figure, I first needed to get the maximum value of a sequence. By using a for loop to sort by the largest number and take the zeroth element I was able to find the maximum value.

```
./collatz -n $n | sort -n | tail -n 1 >> /tmp/maximum.dat
```

I used the sort command on the collatz file to see the maximum value of the sequence; however, while I was doing so, I noticed two things of importance. The first thing I noticed was that it was necessary for me to pipe to less that way I can parse through the file to see the values. Furthermore it was also good for me to note that when the ./collatz | sort | less command is

given, the values generated are shown lexicographically rather than numerically. I wanted to see the maximum value of the sequence, and thus I needed the values to be sorted numerically. By adding -nr after sort, the numbers will sort in numerical order and they will be displayed in reverse, meaning that the maximum number will be at the top rather than at the bottom. Rather than use -nr; however, I choose to use tail which gets the last character of the sequence, in this case that's the maximum value of the collatz sequence.

```
rm -f /tmp/maximum.dat
```

While creating this graph, I once again needed to forcibly clear my data file to make sure that there was a clean slate every time I ran my script. To do so I simply mirrored what I did for the first graph.

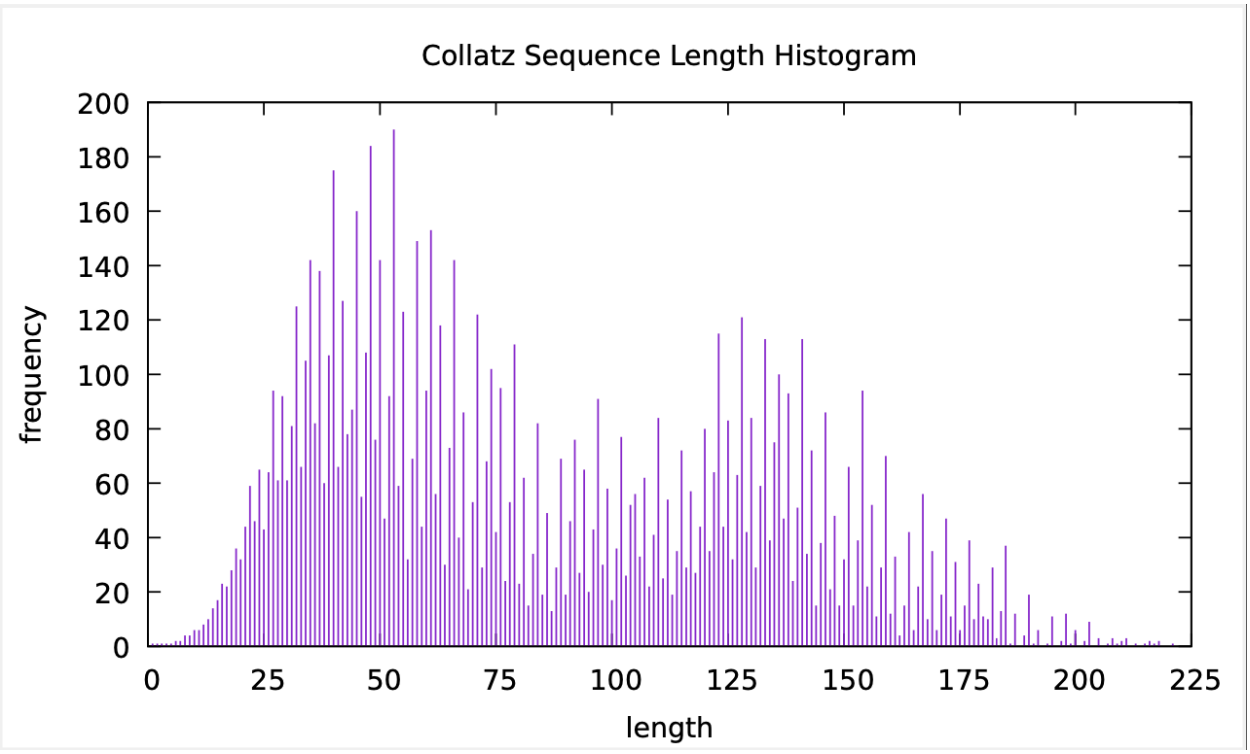
The gnuplot script commands for this figure were also similar to the one given for the collatz sequence length graph, which was modified to fit what was needed for the first figure. In the same way, the gnuplot commands for this figure will be similar to the ones initially given; however, altered out of necessity. My gnuplot commands make it so that the actual plot can be generated by gnuplot. I had the output set to maximum.pdf, the title was “Maximum Collatz Sequence Value”, the xlabel was “n” and the ylabel was “value”. For this graph I also needed to specify the y range by adding “set y range [0:100000]” which extended my y range to what it needs to be. I also instructed gnuplot to also plot the location to maximum.dat with points.

Maximum Collatz Sequence Value (Figure 2) Thoughts:

Overall, this figure was semi easy to create. I struggled with how to sort the values numerically rather than lexicographically, but after watching Eugene's section video and viewing the man page on sort, I was able to find an easy solution to my problem. Another thing I had trouble with

while creating this graph was due to my own carelessness. While setting the yrange in the gnuplot script commands I accidentally added an extra 0, so instead of my yrange being 100,000, it was 10 million. This compressed my graph when I viewed it and skewed all my data. After seeing the values on the y axis label, I quickly realized my mistake and fixed the graph.

Collatz Sequence Length Histogram (Figure 3):



Collatz Sequence Length Histogram was the third and final figure that I needed to generate for assignment 1. This plot is a graphical representation of the collatz sequence lengths in histogram form. I began the shell script commands for this graph in the same way I began the commands for the other graphs I made, using a for loop. The range of this for loop is the same as the range used to create the other two graphs, 2 to 10,000.

To start to create this figure I first needed to place all the sequence lengths into one file in order to see that there were clearly repeating numbers.

```
cat /tmp/length.dat | sort -n | uniq -c | less >> /tmp/histogram.dat
```

In order to create a graph that displays a histogram of Collatz sequence lengths starting from $n \in \{ 2, \ldots, 10000 \}$ I also need to uniq the repeated elements and then plot them. The uniq command in bash reports or filters out repeated lines in a file and it can also precede each output line with the count of the number of times the line occurred in the input if a -c is added

afterwards. By doing this to my length.dat file I was able to see how many times each element is repeated.

I initially used the command cat /tmp/length.dat | uniq -c | less because it was able to show me the exact amount of times an element is repeated in the collatz sequence lengths. I later added sort -n, in order to have the file be sorted numerically. This command specifically concatenates my length.dat file, sorts it numerically, filters and counts the repeated lines and then appends them to a new data file titled histogram.dat.

```
rm -f /tmp/length.dat
rm -f /tmp/histogram.dat
```

Because I am utilizing two data files to generate this graph, there are two data files that needed to be forcibly cleared. I ran into an issue early on while coding the script commands for this figure because I forgot to clear the length.dat data file. Once I realized it and fixed my mistake the issue was resolved.

The gnuplot commands for this figure also remained similar in structure when compared to my other gnuplot commands; however, the contents once again changed slightly. The gnuplot commands for this figure have the output going to “histogram.pdf” and the title set as “Collatz Sequence Length Histogram”. The xlabel was set to “length” and the ylabel was set to “frequency”.

```
set xtics (0,25,50,75,100,125,150,175,200,225)
set ytics (0,20,40,60,80,100,120,140,160,180,200)
```

I also had to manually set the tics on the x and y axis by using the command set xtics and set ytics. I had gnuplot plot the location to histogram.dat using 2:1 (meaning that the x should be column 2 and the y should be column 1, thus inverting the axis) with impulses rather than points.

Collatz Sequence Length Histogram (Figure 3) Thoughts:

Overall, I think that this graph was the most tricky to generate because of the way it was necessary to numerically sort through the data and then identify the repeated elements, rather than simply identifying the recurring numbers. Setting the axis tics manually was also something that took time for me to think of and it was necessary for me to reference the gnuplot.info website to see how it was done. (http://www.gnuplot.info/docs_4.2/node295.html)

Conclusion:

In this assignment I familiarized myself with a UNIX system and learned how to write a bash script. It was really interesting to see how just a few commands in bash were able to generate a detailed graph using the collatz.c program that was provided. My favorite part of this assignment was learning how to troubleshoot the issues I ran into along the way. If something didn't work I would scrap what I had and try again. By doing this I was able to learn a lot more. My main takeaways from this assignment was that

1. The sort command in bash will not numerically sort your elements, but rather lexicographically
2. Commenting out bits of your code to try to find an error is one of the most effective and efficient ways to troubleshoot.
3. It is always good to come up with a game plan for how you want to tackle a code or assignment rather than just jumping straight into it.